Mécanisme prédictif d'évaluation des caractéristiques géométriques des circuits VLSI

Iping Supriana Suwardi

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
«Informatique»

par

SUWARDI Iping Supriana

MECANISME PREDICTIFS D'EVALUATION DES CARACTERISTIQUES GEOMETRIQUES DES CIRCUITS VLSI.

Thèse soutenue le 3 juin 1985 devant la commission d'examen.

L. BOLLIET	Président
F. ANCEAU	
D. ETIEMBLE	
A. GUYOT	Examinateurs
A. JERRAYA	
C. MASSON	

RESUME

Le travail présenté dans cette Thèse porte sur le domaine de l'aide à la construction du plan de masse de circuits VLSI. Cette construction est basée sur une évaluation topologique prédictive et une approche hiérarchisée.

FLOPE est un éditeur interactif permettant la construction d'un plan de masse de manière structurée.

Il est essentiellement destiné à communiquer avec des évaluateurs existants ou à venir.

Son rôle dans la conception hiérarchique est notamment :

- d'anticiper les problèmes de **composition** grâce à l'évaluation prévisionelle de surface, de forme et d'interconnexions lors de l'étape de **décomposition.**

- d'absorber souplement les modifications topologiques grâce à un mécanisme de **propagation.**

FLOPE a été implanté en langage CEYX-Le_Lisp.

Mots-clés :

éditeur hiérarchique, élaboration du plan de masse, évaluateur, cohérence topologique, propagation, voisinage, interface.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M. MERMET
Directeur des Etudes et de la formation : Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique - Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique Industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

Je tiens à remercier :

- Monsieur L. BOILLET, Professeur à l'université de Grenoble II, de l'honneur qu'il m'a fait en acceptant de présider le jury de cette thèse.

- Monsieur F. ANCEAU, Professeur à l'ENSIMAG, pour avoir dirigé cette thèse, et fait l'honneur de faire partie du jury.

- Monsieur B. COURTOIS, Maître de recherche au CNRS, pour m'avoir accueilli dans son équipe et pour les encouragements qu'il m'a prodigué dans mon travail.

- Monsieur D. ETIEMBLE, Professeur à l'Institut de Programmation Université Paris IV, pour avoir accepté de participer au jury.

- Monsieur C. MASSON, Chef du Service CAO-VLSI Bull, qui a bien voulu juger mon travail et participer au jury.

- Monsieur A. GUYOT, pour nos nombreuses discussions qui ont permis l'enrichissement de ce travail, et pour sa participation au jury.

- Monsieur A. JERRAYA, pour les conseils dans mon travail, et pour sa participation au jury.

- Monsieur F. ROUGEAUX, pour sa patience à réviser le français de ce texte.

- Mes collègues, les membres de l'équipe d'architecture des ordinateurs avec qui j'ai eu des échanges fructueux.

- Messieurs D. IGLESIAS et C. ANGUILLE et le Service de Reprographie de l'Institut IMAG, pour l'excellente qualité du tirage effectué.

- Ma famille et mon épouse, RIKE, pour le réconfort qu'elle m'a toujours apporté et la patience dont elle a fait preuve.

TABLE DES MATIERES

INTRODUCTION

La conception de circuits VLSI (Very Large Scale Integration) est une activité dont la difficulté croit avec l'augmentation de la densité d'intégration.

Afin d'affronter la complexité inhérente à la conception d'un circuit VLSI, une approche hiérarchisée et structurée est généralement adoptée [PRE-78], [MEA-80].

Une amélioration importante, basée sur cette approche, consiste à établir dès l'étude de l'architecture du circuit, puis à faire évoluer au fur et à mesure de la conception, un plan général, appelé **plan de masse.**

Selon cette approche, un tel plan correspond à la fois au découpage fonctionnel initial du circuit et au plan général de son implantation.

Nous nous sommes intéressés à des outils de plan de masse. Cette étude est développée dans le cadre du projet de conception des circuits VLSI, **CAPRI.**

Notre travail s'articule autour de deux points :

- L'étude et la réalisation d'un éditeur de plan de masse basé sur les caractéristiques géométriques des blocs constituants des circuits VLSI.

- La gestion d'une interface entre l'éditeur et un ensemble d'outils d'évaluation topologique. Elle permettra l'élaboration d'un plan dont la précision augmente avec l'avancement des activités de conception.

Cette thèse se compose de sept chapitres :

Le chapitre I contient la méthode de conception des circuits VLSI. Il présente le rôle du plan de masse dans le cycle de conception. Il présente également le système CAPRI comme cadre de notre travail.

Le chapitre II contient le principe d'évolution d'un plan de masse, lequel est basé sur la méthode des raffinements successifs.

Le chapitre III décrit l'organisation du plan de masse et le problème principal rencontré dans un cycle de raffinement.

Le chapitre IV est consacré à l'étude des représentations internes d'un plan de masse.

Le chapitre V traite l'interface entre l'outil de plan de masse et les outils d'évaluation topologique (évaluateur).

Le chapitre VI présente la spécification externe de l'outil du plan de masse.

Le chapitre VII décrit l'implantation de l'outil en langage CEYX_LELISP.

Chapitre 1

METHODE DE CONCEPTION DES CIRCUITS VLSI.

1.1. INTRODUCTION

La conception des circuits VLSI a pour but la génération des masques. Elle couvre deux domaines : **logique et topologique.**
La conception logique consiste à traduire en un schéma logique une spécification fonctionnelle donnée pour un circuit.
La conception topologique, consiste à concevoir l'image de l'implantation physique d'un schéma logique sur la pastille de silicium.
L'une des étapes, importante dans le cycle de conception, est la construction du **plan de masse** d'un circuit. Le plan de masse reflète l'implantation physique des blocs qui constituent un circuit.
La complexité actuelle des circuits rend de plus en plus problematique cette étape. Il s'agit de placer et de connecter entre eux plusieurs blocs en réalisant un compromis entre l'optimisation de surface et le temps de conception.

1.1.1.
Limites des schémas classiques de conception des circuits VLSI.

Dans la plupart des méthodes de conception de circuits VLSI (fondées sur le principe schématisé par la figure I-1), la conception logique se fait sans aucune considération des aspects géométriques. Elle est suivi par l'étape de conception topologique. Celle-ci consiste à transformer la description logique en dessin de masque, tout en tenant compte des contraintes géométriques suivantes :

- la forme et la surface des blocs fonctionnels,
- les blocs voisins,
- le routage global entre les blocs.

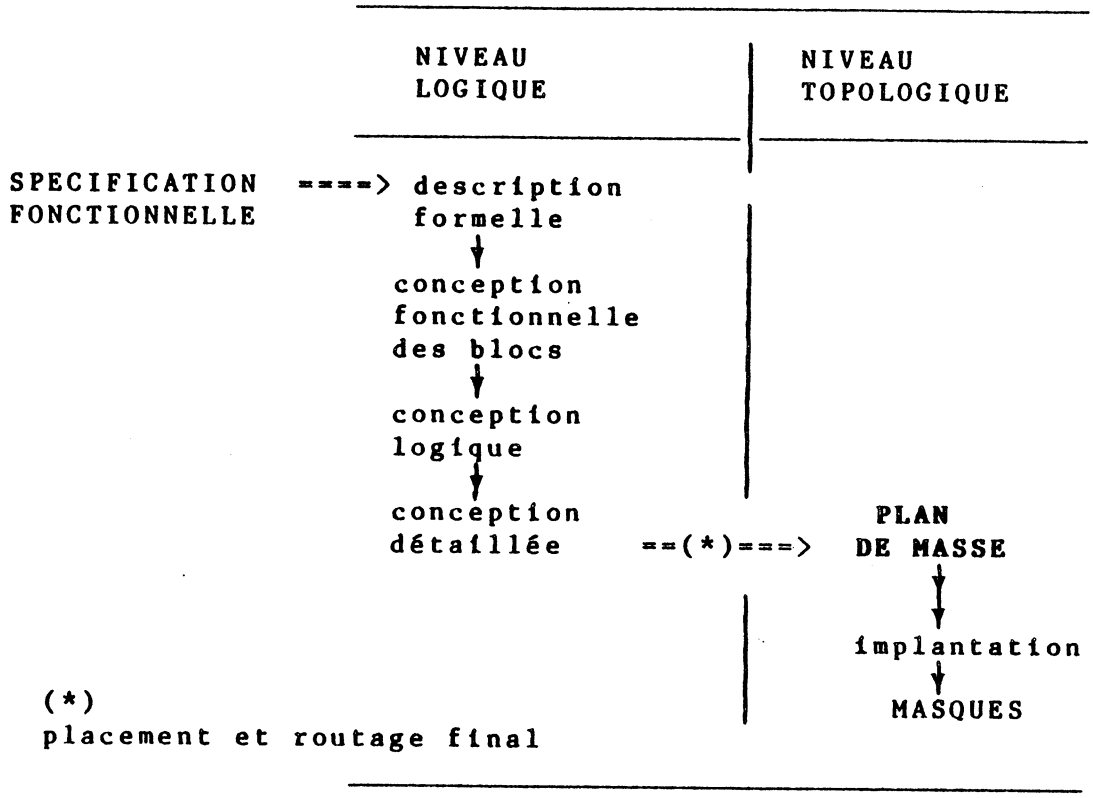


Fig I-1. Schéma de conception conventionnel.

Avec ce type de conception, les contraintes n'apparaissent que dans les dernières étapes, lors de la réalisation d'un circuit. Un processus de fabrication est généralement supporté par des outils de placement et routage. Ces outils servent à établir le plan de masse, à partir des blocs conçus et optimisés indépendement, au moment de la conception détaillée.

Si pour respecter les contraintes topologiques globales, la forme d'un bloc doit être modifiée, alors tout le processus depuis l'aspect logique doit être revu en tenant compte de ce nouveau paramètre et un allongement significatif du temps de conception apparait.
Cet allongemet est notamment causé par le retard de la contré-réaction de l'aspect topologique sur l'aspect logique.

1.1.2. Réduction des cycles de conception.

Face au problème évoqué ci-dessus, plusieurs solutions ont été proposées. Bien que les approches soient différentes, le principe est presque toujours le même. **La construction du plan de masse doit commencer tôt et se développer par étapes, en parallèle avec la conception logique.**

Une proposition de Maling [MAL-82] permet de tenir compte des contraintes géométriques par étapes très tôt dans le processus de conception. Elle est basée sur le fait que les représentations logiques et géométriques peuvent être visualisées comme deux graphes planaires duaux l'un de l'autre. Le plan de masse peut alors être construit à partir du graphe des connexions de faon automatique.

Otten et al. [OTT-82] ont proposé une autre approche basée sur des projections de graphe d'interconnexion sur un plan de deux dimension.

Une autre approche, qui a été proposée par F. Anceau et R. Reis [ANC-82a] consiste à paralléliser la conception logique et topologique en se basant sur la possibilité d'évaluer les caractéristiques topologiques à partir de la décomposition fonctionnelle d'un circuit en fonction des contraintes géométriques.

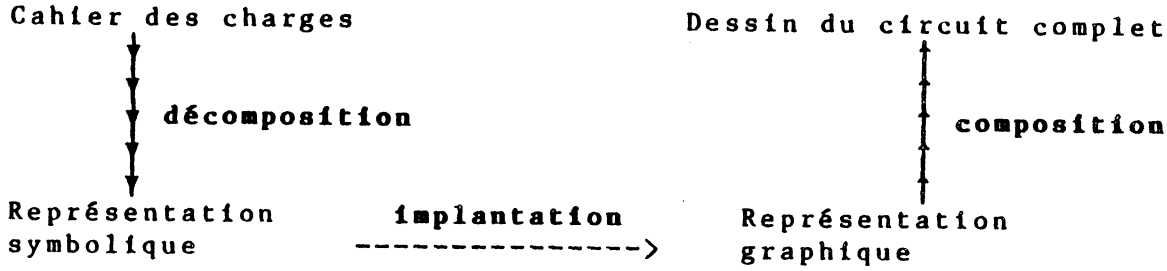
Notre étude n'est pas de chercher une solution au problème évoqué à la section I.1.1, mais de concevoir et de réaliser les mécanismes de construction capable de supporter la dernière approche dans le cadre du projet d'un compilateur de silicium, **CAPRI.** [ANC-82b]

1.1.3. Réduction de la complexité de conception.

Afin d'affronter la complexité inhérente à la conception de circuits VLSI, une approche **hiérachisée** doit être utilisée et l'outil d'aide doit supporter la méthode de conception [PRE-79, MEA-80, SOU-80, SZE-80].
En se basant sur l'approche hiérarchique, la démarche de conception

peut être divisée en trois étapes [GUY-83] :

1. Décomposition (approche descendante)
2. Implantation
3. Composition (approche ascendante).



Décomposition.

Démarche cartésienne de découpage de bloc exécutant une fonction en un ensemble de blocs exécutant chacun une fonction plus simple, mais coopérants pour exécuter la fonction précédente, et d'iteration du découpage jusqu'à ce que les fonctions soient suffisamment simples pour qu'on puisse en entreprendre l'implantation. Au cours de cette étape est fait un choix de "style" (logique anarchique, briques, PLA, etc ...). Le choix du style conditionne :

- le nombre de transistors requis pour exécuter la fonction
- la densité (transistors par surface) prévisible
- la déformabilité d'un bloc
- la facilité d'interconnexions
- l'effort de dessin

Pour connaître tous les termes compromis, il est nécessaire de connaître au plus tôt le plan de masse du circuit.

Implantation.

Consiste à passer d'une représentation variable de la fonction de chaque blocs (schéma logique, schéma à transistors, contenu binaire d'un PLA ou d'une ROM, etc .....) à une représentation unique pour tout le circuit des masques (dessin au micron, au lambda, stick diagramme, ...) et ceci pour chaque bloc.

Composition.

Consiste à passer de la représentation du masque de chaque bloc à la représentation des masques du circuit complet.

Ces trois étapes, idéalement séquentielles, sont polluées par de fréquents retours en arrière ou au contraire des avances prématurées.
En effet, l'implantation d'un bloc (c'est-à-dire son dessin) peut remettre en cause sa fonction (respectivement sa forme) et par effet de domino celle de tous les blocs qui lui sont connectés (respectivement les blocs voisins).

Afin d'anticiper le problème de composition, il est nécessaire d'estimer la taille et forme de chaque bloc [ANC-83a]. Ces estimations sont successivement rafinées.

Cette démarche de conception permet d'assurer, avant de commencer la conception détaillée d'un niveau plus bas dans la hiérarchie, une bonne estimation de l'exigence (taille, surface et position) de module supérieur.

Le résultat d'estimation devient à la fois une guide et une "contrainte" géométrique pour l'implantation et la composition finale.

1.2. CAPRI : Compilateur de Silicium.

Un compilateur de silicium transforme une description fonctionnelle en dessin de masques [AYR-79]. Pour y parvenir facilement, un modèle ou plan de masse standard est utilisé.

Le système CAPRI (Conception Assistée PRocesseurs Intégrés) est un compilateur de silicium mettant en oeuvre la méthodologie de conception CAPRI [ANC-82b], [ANC-83b]. L'objectif du système est de réduire le coût et la durée de conception des circuits VLSI. Sa méthodologie s'applique à tous les circuits qui peuvent être decomposés en une partie opérative et une partie contrôle.

1.2.1. Méthodologie CAPRI.

La conception de circuits VLSI à partir d'une description comportementale en utilisant la méthodologie CAPRI (figure I.2) est divisée en quatre étapes. A chaque étape on fait des choix. Ces choix peuvent être guidés par l'outil d'évaluation.

o **La conception architecturale.**
Elle consiste à déterminer les spécifications des parties opérative et contrôle à partir d'une description comportementale.

o **La conception de parties opératives.**
Elle consiste à générer le dessin des masques d'une partie opérative à partir des spécifications extraites lors de la première étape.

o **La conception de partie contrôle.**
Elle consiste à générer le dessin des masques d'une partie contrôle à partir des spécifications extraites lors de la première étape.

o **La conception finale.**
consiste à compléter les deux générations précédentes par :
- les mécanismes d'interruptions.
- les plots d'entrées-sorties particuliers.
- le système d'horloge.
- ....

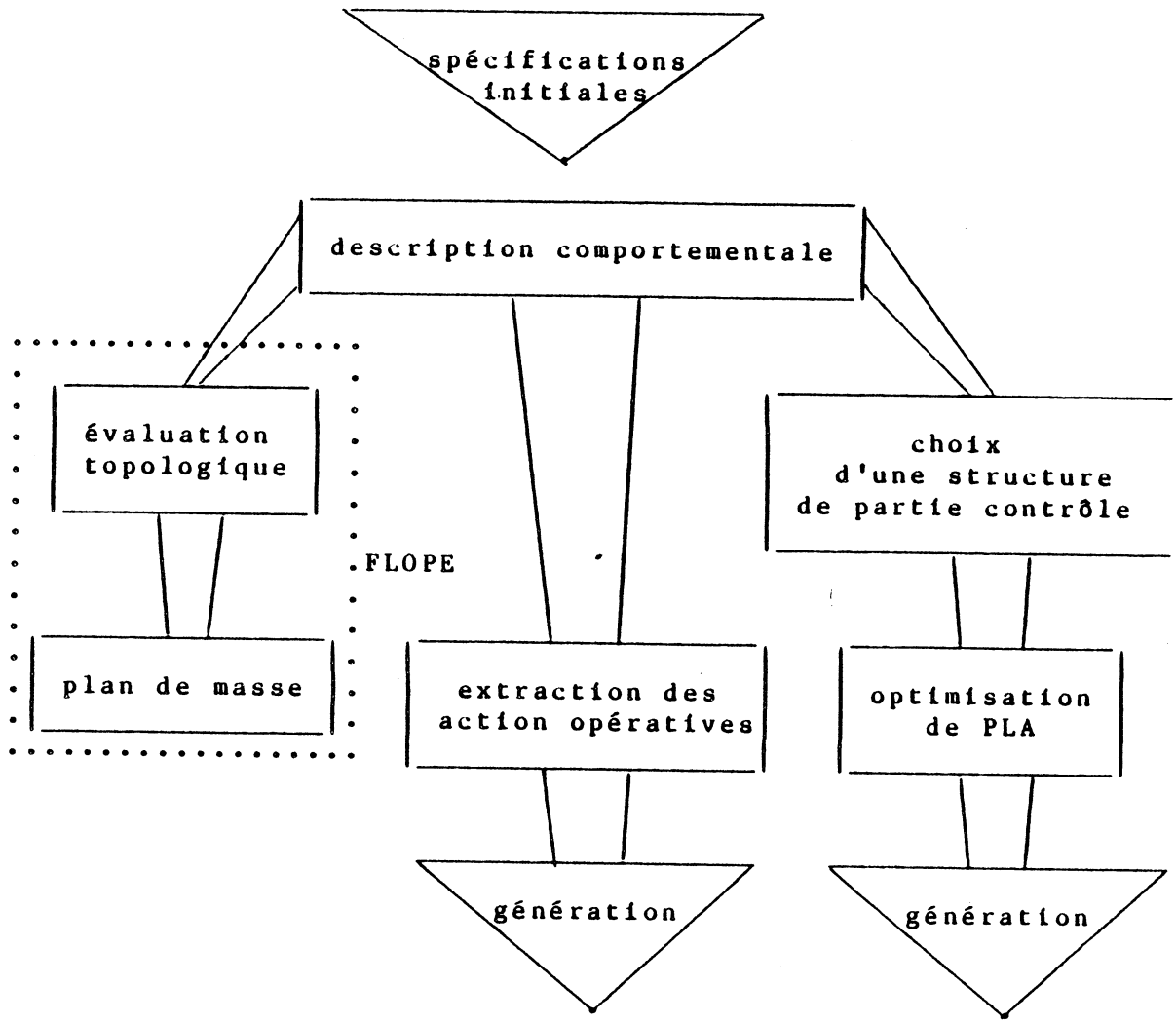


Fig I-2. Schéma de conception selon le méthodologie CAPRI.

Le système CAPRI est constitué d'un langage de description du materiel appelé IRENE, et d'un ensemble d'outils qui permet de passer de la description IRENE au dessin des masques.
Le principaux outils du système CAPRI sont

- Le langage de description IRENE qui permet la description structurelle et comportementale de circuits sans ambiguité [MAR-83]
- Un extracteur qui définit les structures de la partie opérative et de la partie contrôle à partir de la description IRENE [SCH-83]
- Un générateur de parties opératives [SUZ-82]
- Un générateur de parties contrôles
- Un optimiseur de PLA [CHU-84]
- Un assembleur de cellules [SCH-83]

- Un vérificateur des schémas électriques qui génére une description électrique à partir du dessin des masques d'un circuit quelconque [JER-83]
- Un évaluateur topologique [REI-83]
- Notre éditeur de plan de masse FLOPE.

1.2.2. Optimisation topologique dans la méthodologie CAPRI.

CAPRI utilise le plan de masse comme élément de départ pour la conception logique et topologique. L'optimisation topologique visée par la méthodologie CAPRI est une **optimisation globale**, au contraire d'une optimisation de blocs conçus indépendemment.
Elle est fondée sur **les règles** suivantes :

- la forme des blocs qui constituent un circuit VLSI doit être ajustée en tenant compte de leur environnement (les blocs voisins). La modification de la forme des blocs est contrôlée par leur fonction de **déformabilité.**

- l'optimisation des blocs d'interconnexion est obtenue :

  • en choisissant les formes des blocs qui peuvent être directement aboutées.

  • en passant les connexions au dessus d'un bloc si c'est possible. Cette possibilité est mesurée par le degré **transparence** d'un bloc.

1.2.2.1. Transparence.

La transparence d'un bloc est un "espace libre" à l'interieur du bloc permettant de passer un certain nombre de fils de connexion sans changer la dimension du bloc [REI-83].

1.2.2.2. Déformabilité.

La déformabilité d'un bloc, est sa capacité à avoir plusieurs formes différentes. Cette propriété est très importante, pour le choix d'une forme qui tient compte de l'environnement dans lequel sera placé un bloc dans le circuit. L'étude de cette capacité

[MEY-82] a montré que la déformabité d'un bloc registre peut être représentée par un courbe comme dans la figure I-3. Cette courbe est appelée fonction de déformabilité, et la formule générale peut être approchée [ANC-83a] par :

$$dx = K / (dy - delta) - delta$$

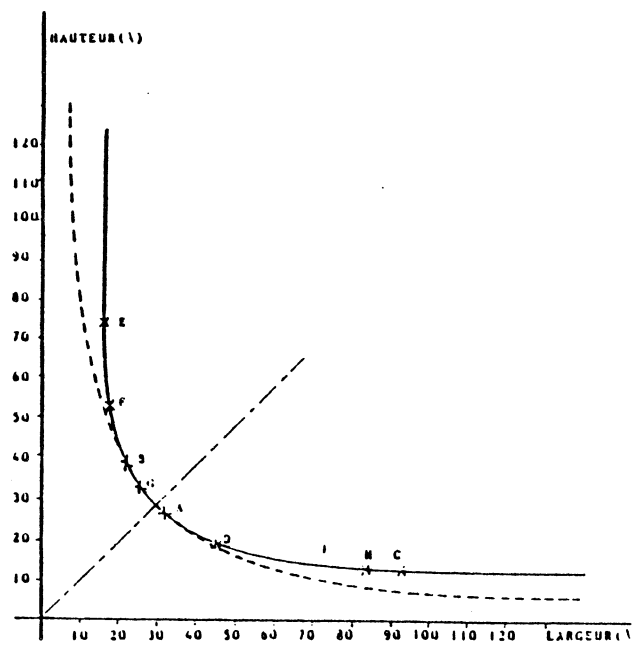


Fig I-3. Courbe de déformabilité d'un bloc registre.

où **delta** est la surface minimale d'un bloc, et **K** un coefficient lié à sa surface minimale calculée par l'évaluateur topologique.

1.2.3. L'évaluateur topologique.

L'évaluateur est un programme permettant de fournir des estimations sur les caractéristiques géométriques : la forme, la surface et la disposition des connecteurs. Cette estimation est basée sur des **paramètres de particularité** des blocs. Par exemple, pour un bloc de type ROM, ces paramètres peuvent être le **nombre de mots**, le **nombre de bits par mot** et les **paramètres technologiques** [REI-83]. Pour un bloc de type LOGIQUE-ALEATOIRE, ces paramètres peuvent être le **nombre de transistors** et les **paramètres technologiques**.

1.3. OUTIL DU PLAN DE MASSE.

La mise en oeuvre de la stratégie d'optimisation topologique globale (cf. 1.2.2) nécessite une organisation et une gestion du plan de masse pour contrôler l'évolution de la conception aux niveaux logique et géométrique afin :

- de définir la topologie globale du circuit, en dégagant très tôt des élements prévisionnels, en faisant appel à des outils d'évaluation topologique.

- de décomposer un problème global en sous-problèmes, en fonction des décompositions fonctionnelles et des connexions entre blocs.

- d'optimiser la surface d'un bloc, en fontion de son environnement.

L'objectif principal de l'étude est donc de concevoir et de réaliser des mécanismes d'aide à la conception du plan de masse, appelé éditeur **FLOPE**, répondant aux critères suivants :

- Le système doit permettre d'accueillir indépendemment les programmes d'évaluation des blocs de circuit VLSI dont les caractéristiques sont diverses.

- Le système doit permettre une élaboration du plan de masse qui facilite l'optimisation topologique.

- Le système doit permettre une utilisation interactive, notamment pour gérer et exprimer les contraintes géométriques.

1.4. ETAT DE L'ART.

Si l'on considére que l'objectif de la construction d'un plan de masse est l'implantation finale des blocs constituant un circuit, on trouve beaucoup de systèmes déjà réalisés.

Nous énumerons de manière non exhaustive plusieurs systèmes existant.

Citons d'abord le système RIOT [TRI-82], un outil interactif qui permet de construire le plan de masse en partant des descriptions des blocs dont la forme et la position des connecteur sont connues.

Le système SHARP [CHI-83], permet de construire automatiquement un plan de masse à partir des blocs réference dans un bibliothèque, chaque bloc étant disponible avec plusieurs formes.

Le système ASTRA [REV-83], possede un outil interactif qui permet de construire un plan de masse à partir d'une description textuelle contenant les blocs à interconnecter.

L'éditeur CAF [LEB-83], permet de construire un plan de masse à partir d'une descriptions contenant des blocs dont les connecteurs sont encore mobiles.

L'éditeur MUSIC du système ARSENIC [BOZ-84], permet de construire un plan de masse et d'utiliser ce plan pour élaborer la conception. La construction est basée sur l'allocation de l'espace du plan pour chaque constituant.

L'éditeur de plan de masse présenté dans cette thèse couvre une étape plus avant que les mécanismes précédents : l'entrée de l'éditeur est constituée d'une description contenant les caractéristiques géométriques de chaque bloc dont sa forme est inconnue.

Chapitre 2

ELABORATION DU PLAN DE MASSE.

La construction du plan de masse d'un circuit VLSI dans la méthodologie CAPRI, est basée sur le principe de la décomposition et de l'évaluation topologique avant de procéder à la conception détaillée.

La décomposition consiste à découper successivement un circuit en blocs, un bloc en sous-blocs, jusqu'à ce que le dernier obtenu puisse être estimé par un évaluateur. Cet évaluateur est un programme permettant de fournir des estimations sur les caractéristiques géométriques : la forme, la surface et la position des connecteurs, en fonction des contraintes de dimension et du niveau de précision souhaité.

2.1. NIVEAU DE PRECISION.

Un bloc peut passer par trois stades de raffinement [ANC-83a] :

Dans le premier niveau, l'état du bloc est appelé "flou", sa forme est contrôlée par la fonction standard de déformabilité et les connecteurs du bloc sont donnés par leur nombre.
Un plan de masse constitué par des blocs de ce niveau sert à définir la topologie globale d'un circuit. Il est raffiné par le deuxième niveau d'évaluation.

Dans le deuxième niveau, l'état du bloc est appelé "mou". Sa forme est déterminée par le choix de son organisation interne. Ce choix permet de localiser les connecteurs. A partir de cette localisation, il est possible d'évaluer la taille du canal de routage entre un bloc et ses voisins et de prévoir le changement de la taille de ces derniers lors de l'utilisation d'une connexion qui les traverse. Donc, ce niveau sert à prévoir l'interaction des connexions entre deux blocs sur le plan de masse, et permet d'obtenir les modifications sur les formes avec une précision plus fine que lors du premier niveau.

Dans le troisième niveau, l'état du bloc est appelé "dur". La forme du bloc ne peut pas être modifiée et ses connecteurs ont des positions exactes. Ce niveau est obtenu à partir des informations du deuxième niveau contenant les contraintes sur les formes et les positions des interconnexions, à l'aide des générateurs de blocs, ou par le dessin des blocs.

Lorsque tous les blocs sont implantés et ont atteint le troisième niveau, le plan de masse obtenu peut être utilisé pour l'assemblage final.

La figure II-1 donne un récapitulatif de ces trois niveaux.

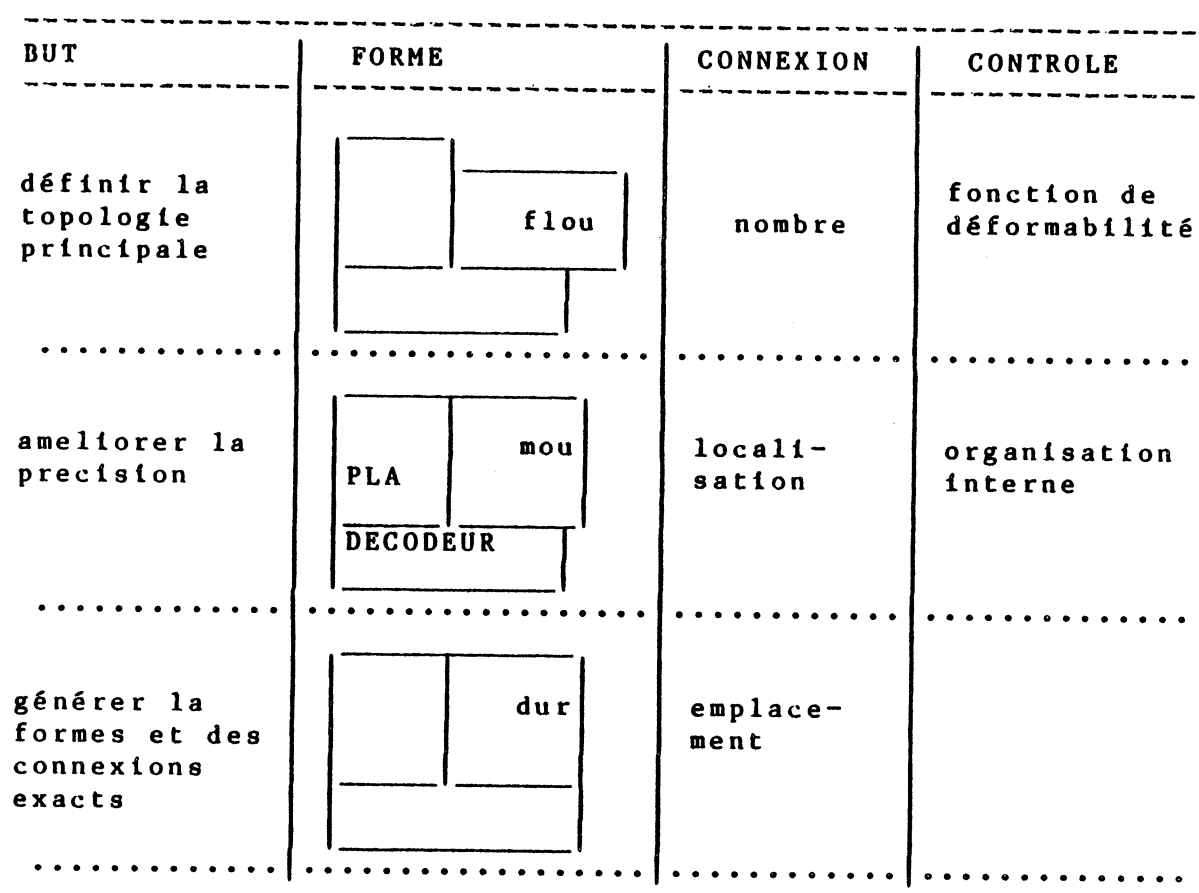


Fig II-1. Récapitulatif du niveau de précision.

2.2. RAFFINEMENT DE BLOCS.

Un raffinement progressif du plan de masse d'un circuit se fait en parallèle avec sa conception logique. A chaque niveau de la conception l'apparition de nouveaux paramètres autorise une évaluation plus fine. Il n'est pas nécessaire d'atteindre une conception détaillée pour commencer. Des premiers raffinement du plan de masse peuvent être effectués dès les premiers niveaux de conception.

En se basant sur les niveaux de précision donnés à la figure II-1, le processus de raffinement d'un bloc en fonction des contraintes peut être schématisé par la figure II-2.

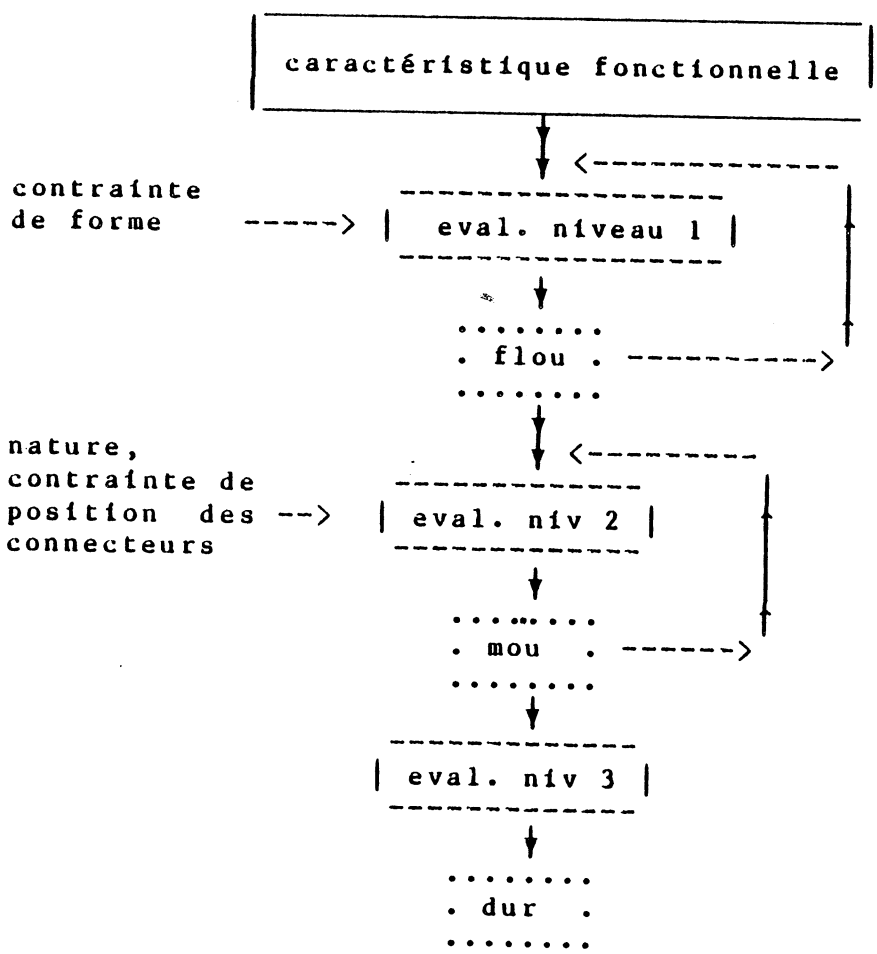


Fig II-2. Schéma des raffinements successifs d'un bloc.

2.3. CYCLE DE CONSTRUCTION.

Partant de la décomposition fonctionnelle d'un circuit et des caractéristiques de chaque bloc, ce cycle peut être divisé en plusieurs étapes. Il faut d'abord évaluer un bloc, ensuite décider du lieu où il sera placé. Enfin, lorsque plusieurs blocs sont déjà placés, il reste de trouver une configuration adéquate.

Ce cycle peut être schematisé comme le montre la figure II-3.

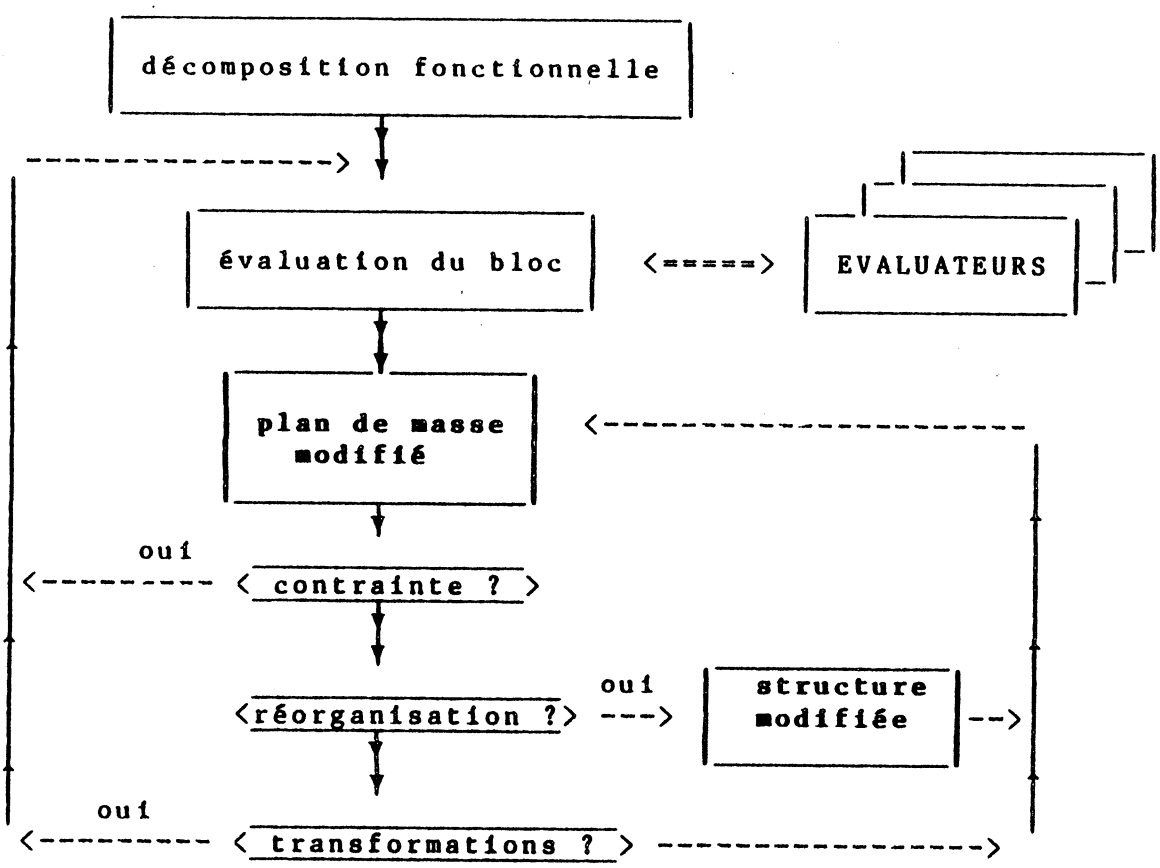


Fig II-3. Cycle de construction du plan de masse.

En se basant sur ce cycle, nous essayons de réaliser l'ensemble des mécanismes nécessaires, et nous étudions un aspect important du plan de masse : sa **représentation interne**, et l'implantation de mécanismes qui en découlent.

Ce cycle fait apparaître trois aspects du problème :

- La structure de données représente :
  - la relation topologique entres les blocs.
  - la structure hiérarchique d'imbrication des blocs.

- Une programme interactif est utilisé pour dialoguer avec l'utilisateur, afin de :
  - placer, retirer, déplacer des blocs.
  - donner des contraintes topologiques que l'utilisateur applique aux blocs.
  - donner l'informations sur les canaux d'interconnexions.

- un ensemble de programmes de prédiction qui comprend :
  - un fonction standard pour évaluer la taille et la déformabilité des blocs au niveau flou.
  - des programmes de prédiction spécifiques aux blocs decident une organisation interne convenable pour chaque bloc, et respectent les contraintes données.
  - les programmes de génération de blocs qui calculent la taille et l'organisation exacte des blocs.

Chapitre 3

ORGANISATION DU PLAN DE MASSE.

3.1. DEFINITION.

Le plan de masse est un ensemble de blocs soumis à une organisation fonctionnelle et topologique.

3.1.1. Organisation fonctionnelle.

L'organisation fonctionnelle représente le découpage fonctionnel du circuit. Dans la méthodologie CAPRI, ce découpage est une conséquence de la conception descendante du circuit.
Ce découpage conduit naturellement à la représentation hiérarchisée des blocs.

3.1.2. Organisation topologique.

L'organisation topologique représente le positonnement des blocs dans le plan de masse. Essentiellement ces blocs sont définis par le nom du bloc, sa position (coordonnées) et son attitude (transformations géométrique qui lui sont appliqués lors de son placement)

3.1.3. Evaluation de la taille d'un bloc.

A un stade donné de la conception, les blocs constituent un circuit intégré peuvent se décomposer en blocs "durs" qui doivent être entièrement remplacés si on veut changer leur forme, et en bloc "mous", dont la forme peut être modifiée, au détriment de la surface, si on change leur organisation interne.

3.1.4. Surface libre.

La surface du plan de masse peut être représentée par un rectangle qui englobe l'ensemble des blocs qu'il contient. La **surface libre**, contenue, à l'intérieur du rectangle englobant, sépare les blocs les uns des autres (figure III-1). Au moment du routage, certaines parties de la surface libre seront allouées aux interconnexions entre les blocs.

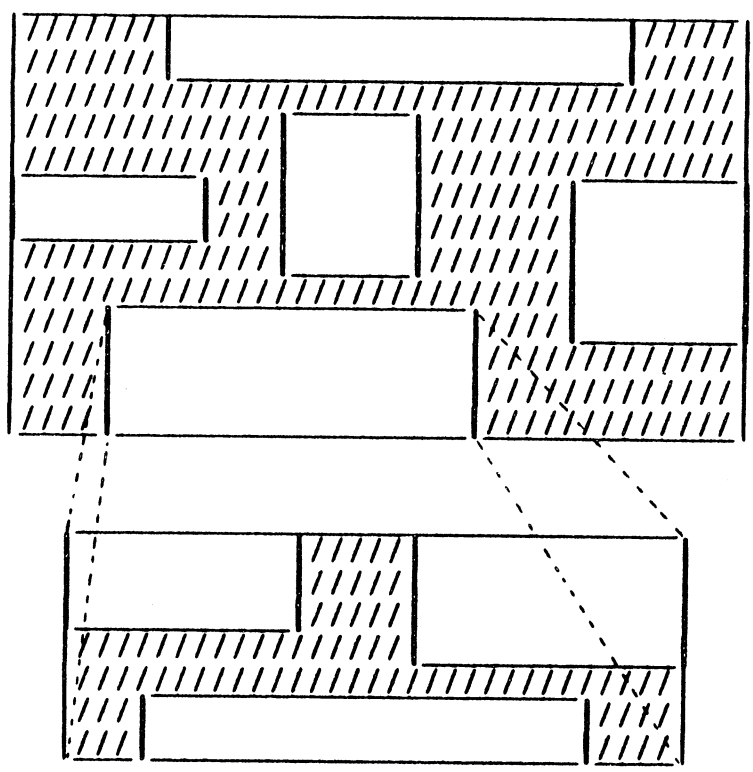


Fig III-1. La surface libre (hachurée).

La tâche d'optimisation du plan de masse consiste notamment à minimiser la surface libre. Une surface minimale d'un plan de masse s'obtient par une suite d'essais (cf. chapitre II). Le mécanisme à étudier ne permet pas de trouver une surface optimale. Il facilite la possibilité d'optimiser la topologie d'un circuit.

3.2. PROBLEME DE L'AJUSTEMENT DE LA FORME DES BLOCS.

L'élaboration du plan de masse que nous avons présenté au chapitre II, a pour objectif d'obtenir une optimisation globale de la surface du circuit en modifiant la forme des blocs, ce qui peut causer des "désoptimisations" locales. La modification de la forme d'un bloc peut entraîner un **chevauchement** entre blocs. Il faut alors déplacer les blocs chevauchés pour éviter les **violations des règles de dessin.**
Dans la figure ci-dessous la modification de la forme du bloc X va entraîner un changement de position pour d'autres blocs.

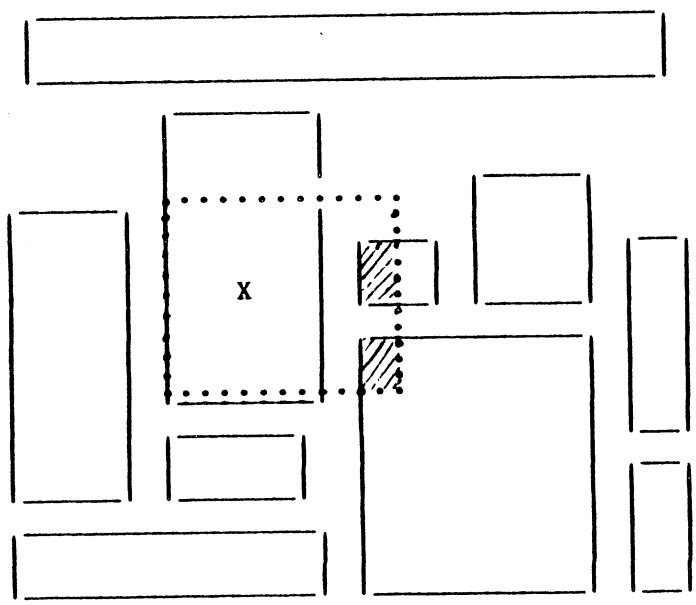


Fig III-2. Un exemple de violation causée par la modification de la forme du bloc X.

On pourrait laisser l'utilisateur résoudre une telle situation. C'est-a-dire, qu'il déplace les blocs chevauchés dans la direction appropriée. Cependant, dans le cas où il y a plusieurs blocs à déplacer, cette activité devient lourde et longue.
Nous pensons que cela doit faire partie des actions du système.

Pour aborder ce problème, dans la section suivante nous allons définir d'abord les notions de **configuration** et de **cohérence.** Ensuite nous introduirons les notions de **propagation, voisinage** et **zone libre,** sur lesquelles le chapitre IV sera fondé.

3.2.1. Notions de configuration et cohérence.

Configuration. On utilise le terme **configuration** pour désigner les positions des blocs dans le plan de masse à un instant donné.
Cohérence. On dit qu'une configuration est **cohérente**, s'il n'y a pas de chevauchement entre les blocs.

3.2.2. Problème principal.

Le mécanisme supportant l'optimisation globale a notamment pour tâche d'assurer la cohérence de la configuration. Il intervient aprés chaque opération topologique, non seulement aprés les opérations de modification de formes, mais aussi pour les opérations d'insertion, de translation, la rotation de blocs, etc.

3.2.3. Environnement des blocs.

3.2.3.1. Propagation.

Pour assurer la cohérence d'une configuration, il est nécessaire de définir un mécanisme qui va permettre de propager l'évènement survenant à un bloc sur les blocs voisins.
Nous appelons ce mécanisme la **propagation.**

Lors d'une modification, la dimension d'un bloc peut être augmentée ou diminuée. Pour cela on envisage deux types de propagation l'une **positive** et l'autre **négative.**

Lorsque la dimension augmente, alors les blocs voisins sont poussés dans la direction correspondante. Pour les déplacer on utilise le mécanisme de **propagation positive.**

Lorsque la dimension diminue, alors les blocs voisins sont tirés vers le bloc modifié dans la direction correspondante. Pour les tirer on utilise le mécanisme de **propagation négative.**
Le mécanisme de propagation négative permet de récuperer la nouvelle région vide.

3.2.3.2. Voisinage.

Pour permettre les propagations, il faut que les blocs aient la "connaissance" de leur environnement. Pour cela on distingue quatre types de voisinages pour un bloc : **nord, est, sud** et **ouest**. La limitation aux quatre directions est suffisante car on ne traite que les formes orthogonales.

3.2.3.3. Zone libre.

Un concept qui nous parrait important, dérivé de la notion de voisinage, est celui de la **zone libre** d'un bloc. C'est un zone qui couvre la surface libre autour d'un bloc (voir la figure III-3). Il est évident que chaque bloc a une zone libre qui lui est propre. La caractéristique intéressante d'une zone libre, est que si la position et la forme d'un bloc modifié ne provoque pas un débordement de sa zone libre, la configuration reste cohérente, sans que l'on ait besoin d'appliquer une propagation.

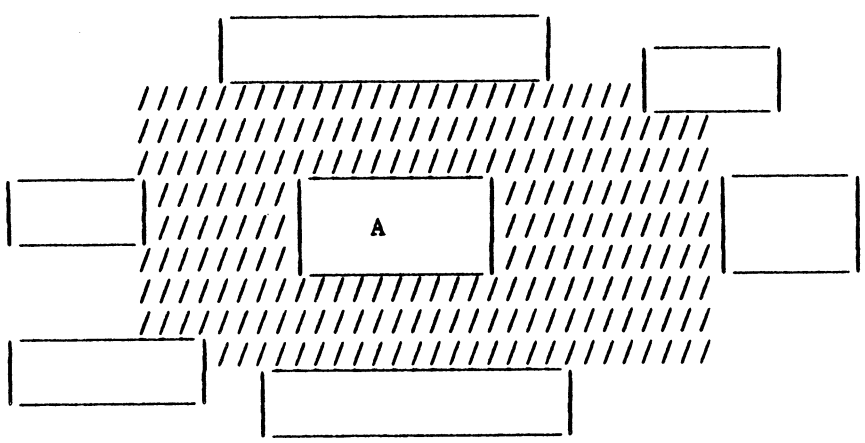
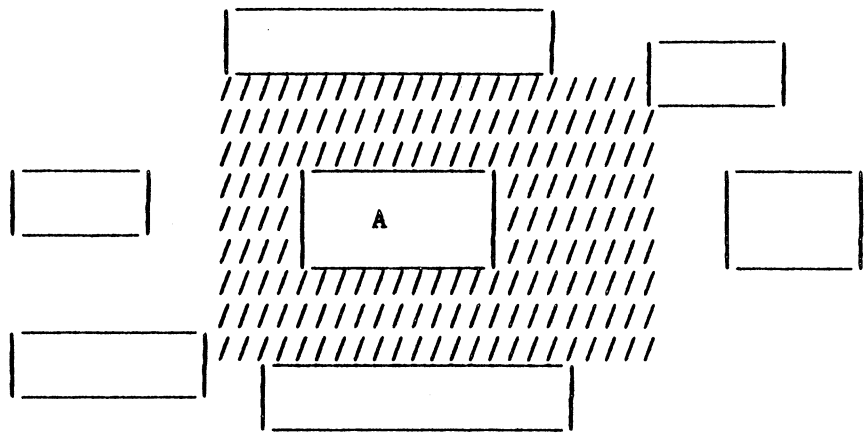
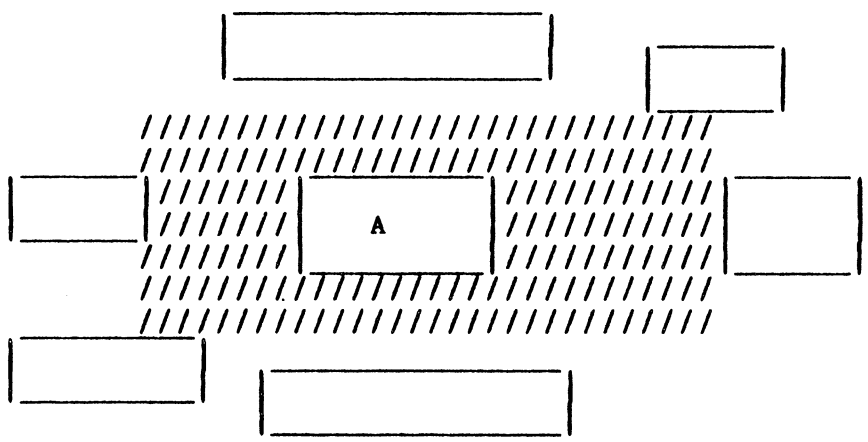


Fig III-3. Zone libre du bloc A.

Idéalement, la forme d'une zone libre est polygonale. Malheureusement, sa détermination et sa manipulation seront très côuteuses. On l'approche avec une forme rectangulaire. La consequence de cette approche est que la forme d'une zone libre n'est pas unique. Selon la modèlisation de voisinage utilisée, cette forme peut être différente pour un bloc donné (voir figure III-4).



(a)



(b)

Fig III-4. (a) une forme approchée
(b) une autre forme approchée

Pour ce qui suit on notera les bords de la zone libre autour d'un bloc par **ZN**, **ZO**, **ZE** et **ZS**. Ces derniers désignent respectivement la frontière au nord, à l'ouest, à l'est et au sud du bloc vide.

Chapitre 4

REPRESENTATION INTERNE DU PLAN DE MASSE.

Dans ce chapitre nous étudions plusieurs modèles permettant de modéliser la relation de voisinage du bloc sur son environnement afin d'assurer la cohérence de la configuration.

Une stratégie pour modéliser cette relation repose généralement sur un découpage de la surface libre [DON-80, SAH-80].

On appellera **découpage de la surface libre** une opération qui consiste à décomposer cette surface en un ensemble des surfaces élémentaires que nous appellerons **objets invisibles.**

La représentation interne du plan de masse représente des **liaisons** établis entre des objets visibles (bloc fonctionnel) et des objets invisibles. Plusieurs types de représentations sont possibles suivant le découpage de la surface libre et l'établissement des liaisons.

La première partie du chapitre est consacrée aux critères jugés principaux dans le choix d'une représentation interne. Dans la deuxième partie nous présentons plusieurs choix possibles. Et dans la troisième partie nous essayons de discuter ces représentations en vue d'obtenir une représentation adaptée à notre besoin.

4.1. CRITERES DE CHOIX D'UNE REPRESENTATION.

Il parrait à priori difficile de concevoir une représentation satisfaisant les besoins de manipulation, existant où à venir, d'un éditeur de plan de masse. Chaque manipulation sur les blocs ne traite qu'une partie des informations données. Les opérateurs de manipulation définis précédemment seront plus au moins adaptés au modèle de représentation choisi.

Nous pensons que cette représentation doit tenir compte de :

- la complexité des mécanismes de base
- la facilité d'estimation des canaux de routage.

4.1.1. La complexité de mécanismes de base.

Les mécanismes importants dans un système interactif sont la création, la suppression et la recherche d'objets. L'expansion de la création et la suppression permettent de construire les autres mécanismes comme la rotation, la symétrie, la translation etc.

Création d'un bloc.

Dans notre problème, la première étape dans la création d'un nouveau bloc est d'évaluer sa taille. Un programme d'évaluation associé au type du bloc va l'évaluer en fonction des paramètres de particularisation donnés.
Cette étape est indépendante de la représentation interne.

Dans la deuxième étape, on détermine les espaces libres disponibles dans lesquels l'utilisateur veut positionner le bloc. On a donc besoin d'un algorithme de recherche des espaces libres. Lorsque l'espace disponible ne peut pas contenir ce bloc, il y a deux possibilités; soit refuser le positionnement, soit l'accepter, la conséquence étant le déplacement d'autres blocs (application de la propagation positive).

Dans la troisième étape, le bloc est inséré dans la structure de données. A cet occasion la relation de voisinage est mise à jour.

La complexité de ces deux dernières étapes varie selon le modèle de représentation interne choisie.

Suppression d'un bloc.

La suppression ne provoque pas d'incohérences sur la configuration. La relation de voisinage doit être mise à jour. Elle peut provoquer des incohérences sur la représentation interne elle même.

L'espace libre laissé par un bloc supprimé doit être récupéré (application de la propagation négative). Cette récuperation peut être faite automatiquement ou à la demande de l'utilisateur.

Modification d'un bloc.

Les modifications peuvent être divisées en deux catégories : les modifications de forme et les modifications d'attitude (par exemple, la rotation d'un bloc est une modification d'attitude). La modification de forme est une action importante dans notre système de prédiction (cf. chapitre II). La représentation interne doit en tenir compte.

4.1.2. L'estimation des canaux de routage.

Le problème du routage consiste à interconnecter les blocs. Le processus de routage peut être divisé en deux phases : **routage global** et **routage effectif** [PRE-79, HOR-81]. Le routage global consiste à décider quels seront les canaux empruntés par chacune des interconnexions. Le routage effectif consiste à implanter les interconnexions sous leur forme définitive sur chaque canal. Dans notre étude la notion de routage ne concerne pas la détermination de trajets exacts, mais plutôt l'estimation de la taille des blocs contenant les canaux d'interconnexion (phase routage global).

La taille de canaux de routage devra être dimensionée de façon à ce qu'elle puisse contenir toutes les interconnexions et/ou tous les fragments d'interconnexions qui lui devront l'emprunter.

Pour que cette estimation soit possible, la représentation interne doit permettre de représenter explicitement la notion de canal.

Principe d'estimation du canal de routage.

Dans un canal (ou bloc) de routage on trouve trois types d'interconnexions : **interne, externe et traverse.**

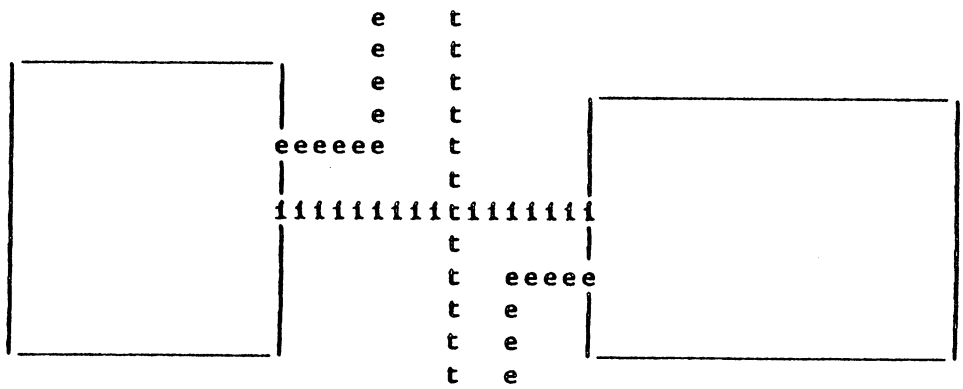


Fig VI-1. Types des interconnexions.

Les connexions internes relient deux blocs voisins. La connexion externe intéresse un seul des deux blocs. La connexion "traverse" est celle qui traverse l'espace entre les deux blocs mais n'appartient à aucun des deux blocs.

L'estimation de canal a largement été étudiée [MAL-81, SAU-84]. Pour chaque type d'interconnexions on associe une valeur, appelée **valeur poids.** A partir des informations sur le type de l'interconnexion et de la technologie utilisée, il est possible d'approcher la dimension des canaux de routage.

La démarche d'estimation se fait alors en deux étapes :

- détermination du trajet : consiste à trouver pour chaque interconnexion un ensemble des canaux que nous appellerons **chemin topologique.**
- affectation de valeur poids pour chaque canal dans un chemin topologique.

Les valeur poids accummulées dans une canal déterminent sa largeur.

4.2. REPRESENTATIONS ETUDIEES.

Dans ce paragraphe nous étudions plusieurs représentations possibles qui permettent de modéliser la relation de voisinage des blocs dans un plan de masse. Ces représentations sont obtenues en appliquant différentes opérations de découpage sur la surface libre.
Pour ces découpages on applique des lignes de "style" :

- guillotine (représentation par interface)
- ciseaux (représentation par séparateur)
- lignes paralléles (représentation par tuile)
- rien (représentation par liste)

Dans chaque répresentation nous décrivons d'abord le principe et la terminologie utilisée. Ensuite nous explicitons la relation de voisinage dans la représentation. Enfin nous abordons les mécanismes de base et essayons de mesurer leur complexité.

4.2.1. REPRESENTATION PAR INTERFACE.

4.2.1.1. Introduction.

4.2.1.1.1. Principe.

On appele découpage par guillotine une opération qui consiste à séparer un ensemble de blocs par une ligne dans un plan de façon à ce que cette ligne dépasse le rectangle d'encombrement des blocs et ne coupe aucun bloc. Cette opération peut se répéter aux sous-plans obtenus en utilisant une ligne ou des lignes perpendiculaires à la ligne précédement utilisée.
La figure IV-2 montre un exemple de plan coupé suivant cette méthode.

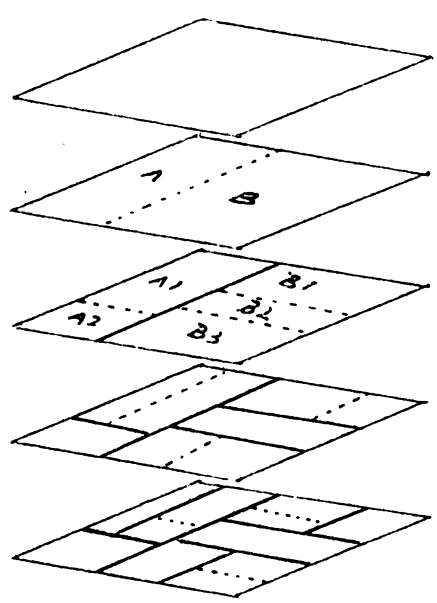


Fig IV-2. Découpage successifs à l'aide des guillotines verticale et horizontale.

La "trace de guillotine" joue le rôle d'objet invisible. Elle est appelé **interface**.

L'interface entre blocs détermine dans le plan les relations de voisinage (**nord**, **ouest**, **sud** et **est**). Par exemple, le voisin au

nord du bloc A2 est A1. Le voisin à l'est du bloc A est B.

Lorqu'un plan est séparé par une interface, les sous-plans obtenus sont appelés la "descendance". Tous les descendants d'un bloc héritent des relations que possèdent leurs ascendants. Par exemple, les blocs B1 ,B2 et B3 sont voisin-est du bloc A2.
Ceci implique que, lorsque la largeur du bloc A2 s'agrandit au moment d'une modification, tous les descendants de B sont poussés. Inversement lorsque la largeur de bloc A2 diminue , tous les descendants de B sont tirées.

L'organisation topologique utilisant la représentation par interface forme un arborescense. Il y a donc deux hiérachies l'une topologique et l'autre fontionnelle (figure IV-3).

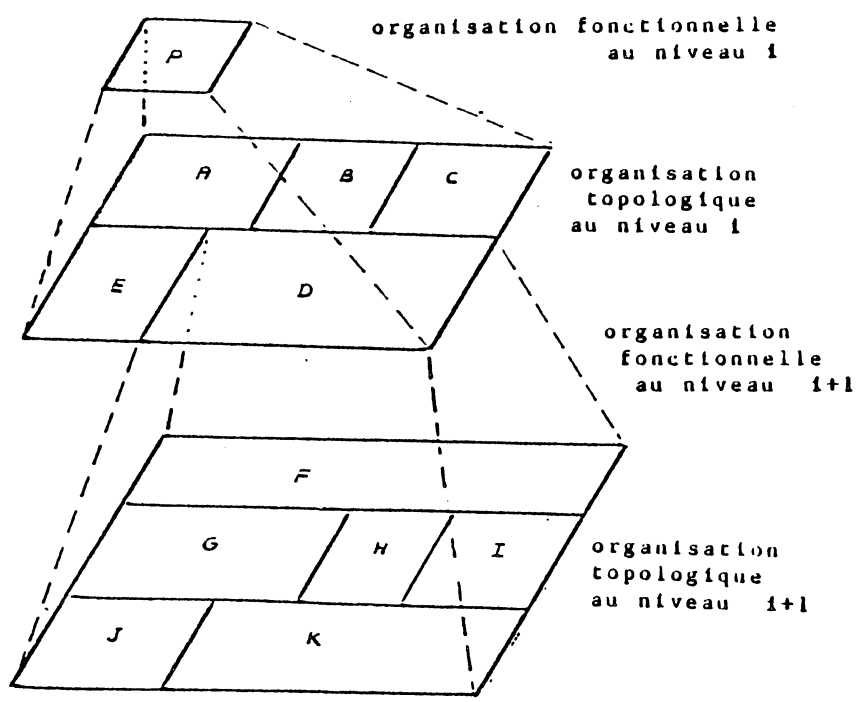
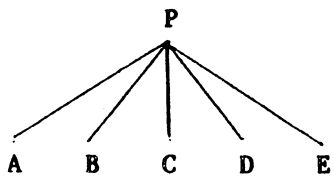
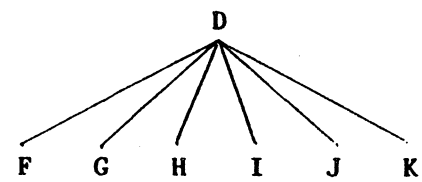


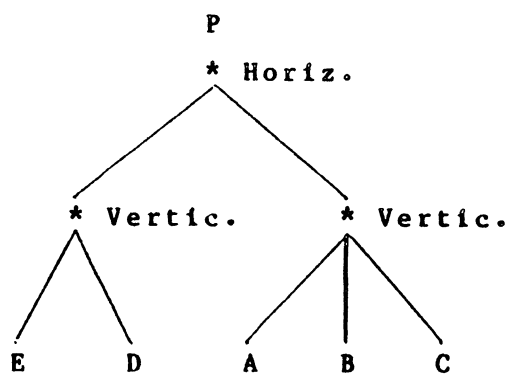
Fig IV-3. Organisation fonctionnelle et topologique du plan de masse.

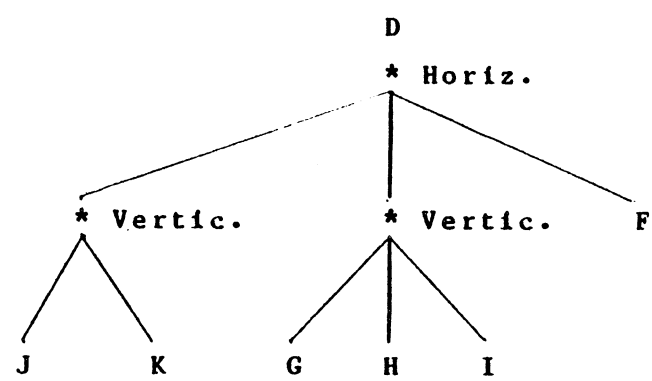
l'organisation fonctionnelle de P et D est :





l'organisation topologique de P et D peut être décrite comme suit :





Dans cette représentation, on peut constater que les deux hiérachies peuvent être unifiées de façon homogène. C'est-a-dire, il n'y a qu'une seule structure qui représente à la fois l'organisation fonctionnelle et topologique.

4.2.1.1.2. Terminologie.

Tous les blocs sont entourés par quatre interfaces, sauf ceux qui sont aux coins (ils en ont 2 au plus) et au bord (ils en ont 3 au plus).

Les blocs et les interfaces sont liés de la façon suivante : Un bloc pointe sur l'interface qui le sépare. Et un interface pointe sur l'interface précédement utilisé.
Dans la structure de données ces liaisons sont réalisés par trois pointeurs. Ces sont les pointeurs père, fils et frère.

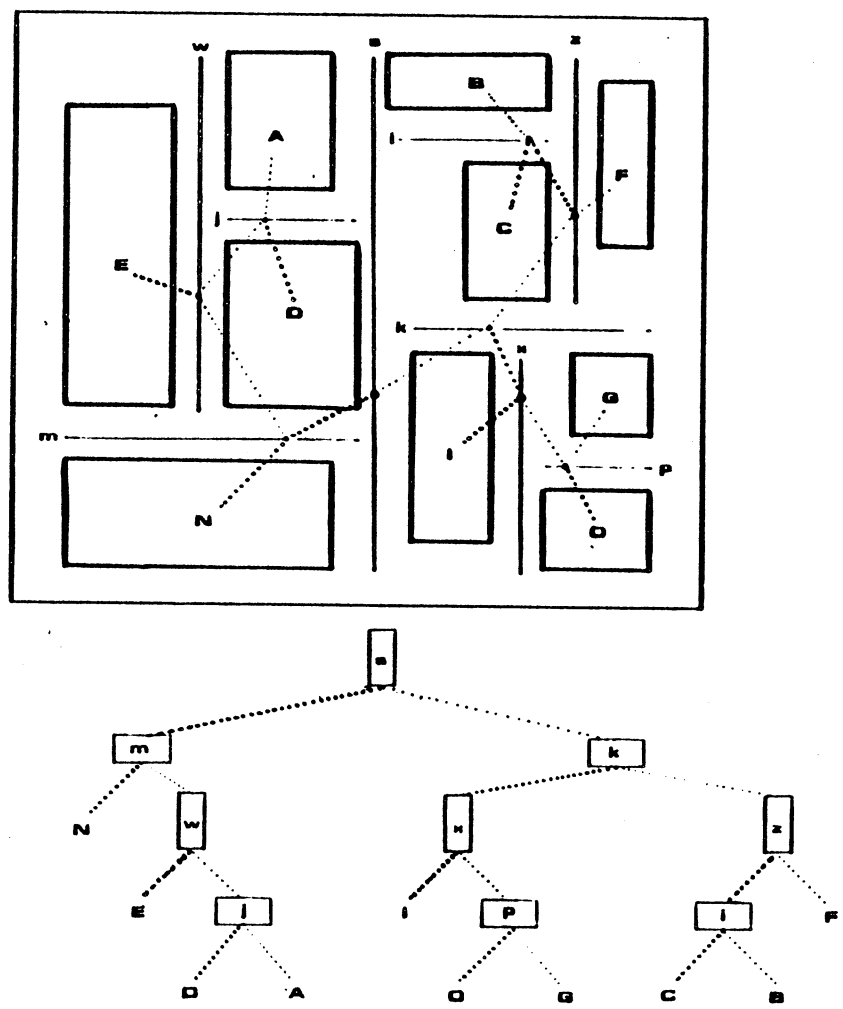


Fig IV-4. Arbre topologique.

On appelle **fils** d'un noeud, tous ses descendants directs. Par exemple, les fils du noeud s sont m et k, les fils du noeud x sont I et p.

Le terme **père** est utilisé pour désigner l'ascendant direct d'un noeud. Par exemple, le père du noeud D est w.

Ensuite, on appelle **frère** d'un noeud, les noeuds qui ont le même père. On distingue les noeuds frères par la relation : **frère-précédent** et **frère-suivant**. Par exemple, le frère-précédent du noeud A est D. Le frère-suivant du noeud D est A.

On utilise le terme **chemin** à un noeud, pour désigner l'ensemble des noeuds situés entre ce noeud et le noeud racine. Par exemple, le chemin au noeud C est C, i, z, k et s.

Pour simplifier l'écriture dans les sections qui viennent, on fait les abréviations suivantes :

PE (b) : père de noeud b.
FI (b) : fils de noeud b.
AV (b) : frère-précédent de noeud b.
SU (b) : frère-suivant de noeud b.
CH (b) : chemin au noeud b.
horizontale (b) : une fonction booléenne qui retourne vrai si le noeud b est une interface horizontale.
verticale (b) : une fonction booléenne qui retourne vrai si le noeud b est une interface verticale.

4.2.1.2. Relation de voisinage.

4.2.1.2.1. Détermination des voisins.

Les voisins d'un bloc sont l'ensemble des blocs qui sont topologiquement situés au nord, à l'ouest, au sud ou à l'est du bloc. Ces voisins sont déterminés à partir de l'arbre topologique.

Voisin nord.
Les voisins nords sont obtenus en remontant l'arbre topologique jusqu'à l'ancêtre étant le fils du premier noeud **horizontal** rencontré. Ils sont l'ensemble des **frères-suivant** de cet ancêtre. Ils sont formalisés de la manière suivante.

$$VN(b) = \{ e \mid e=SU(z),\ z \in CH(b) \text{ et } horizontale(PE(z)) \}$$

Voisin sud.

$$VS(b) = \{ e \mid e=AV(z),\ z \in CH(b) \text{ et } horizontale(PE(z)) \}$$

Voisin est.

$$VE(b) = \{ e \mid e=SU(z),\ z \in CH(b) \text{ et } vertical(PE(z)) \}$$

Voisin ouest.

VO (b) = { e | e=AV(z), z ∈ CH(b) et verticale(PE(z)) }

4.2.1.2.2. Détermination de la zone libre autour d'un bloc.

Dans la section 3.2.3, nous avons noté que les bords de la zone libre d'un bloc sont ZN, ZO, ZS et ZE. On définit les fonctions suivantes pour calculer ces zones.

ZN(b) :

choix :

- null (PE(b)) ====> "infini"
- **horizontale** (PE(b)) et exist (**SU**(b)) ====> **haut** (PE(b))
- vrai ====> ZN (PE(b))

finchoix.

Les autres bords sont déterminés de la même façon. Le tableau ci-dessous indique les fonctions utilisées pour tous les bords.

ZN	horizontale	SU	haut
ZO	verticale	AV	gauche
ZS	horizontale	AV	bas
ZE	vertical	SU	droite

4.2.1.3. Mécanisme de propagation.

En principe, la propagation consiste à ajouter ou diminuer une valeur aux coordonnées des blocs voisins en fonction de la déformation d'un bloc considéré. Cette valeur est appelée "valeur propagée".

Propagation dans la direction nord.

Soit VN(b) les voisins nord du bloc modifié, b. Considérons YVN(b), un sous-ensemble de VN(b) qui subit une propagation causée par la modification du bloc b. Cet ensemble peut être décrit comme suit :

$$YVN(b) = \{\ e \text{ dans } VN(b) \mid Y(e) = Y(e) + p(dy)\ \}$$

où Y(e) désigne l'ordonnée du noeud e et p(dy) est une valeur calculée en fonction de la valeur propagée et de la limite de la zone libre.
Puisque la détermination des voisins et celle de la zone libre nécessitent un parcours à partir du bloc modifié jusqu'à la racine, la propagation consiste elle aussi, en un seul parcours en comparant l'encombrement du noeud visité et celui des voisins à déplacer.

L'algorithme suivant décrit la propagation dans la direction nord.

```
proc PN(b,VP)  { b: bloc, VP: valeur propagée }
  si VN  = 0 alors termine.
  sinon
    . si horizontal(père(b)) alors
          déplacer les voisins-nord du b au long de VP
    . PN( père(b), nouvel_hauteur(père(b)) - hauteur(père(b)) )
```

où nouvel_hauteur est une fonction qui calcule la hauteur de l'ensemble des blocs fils après déplacement.

Le temps nécessaire à la propagation est proportionnel au nombre de fils de chaque noeud dans le chemin du bloc modifié. Dans une arborescene dont les branches sont balancées, la complexité de cet algorithme est inférieur à la moitié du nombre de noeuds existants. Tandis que dans le pire cas (par exemple, l'arbre binaire étant une liste) la complexité peut être égal au total des noeuds existant.

La propagation dans les autres directions est réalisée de la même façon.

A titre d'exemple, la figure ci-dessous montre une propagation causée par la modification du bloc X. Les blocs A, B, C, D et E subissent une propagation positive, tandis que le bloc F subit une propagation négative.

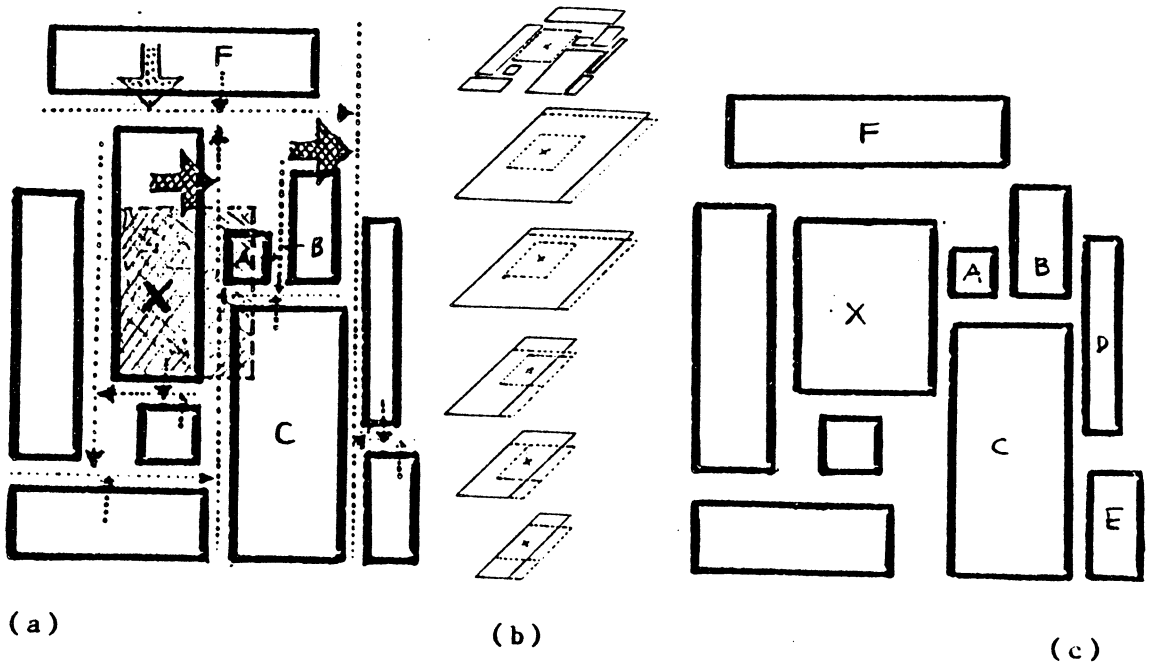


Fig IV-5. Exemple de propagations est et sud.
(a) configuration avant la modification du bloc X
(b) séquence de propagation (de bas en haut)
(c) configuration après la modification du bloc X

4.2.1.4. Mécanismes de base.

4.2.1.4.1. Recherche d'un point.

La recherche d'un point a pour but de sélectionner les blocs se trouvant à un point donné.

L'algorithme de recherche commence en partant de l'interface racine. Selon la position du point donné par rapport à l'interface parcouru, la recherche continué à droite, à gauche, en haut ou en bas sur une interface ou un bloc. La figure IV-6 montre cette recherche.

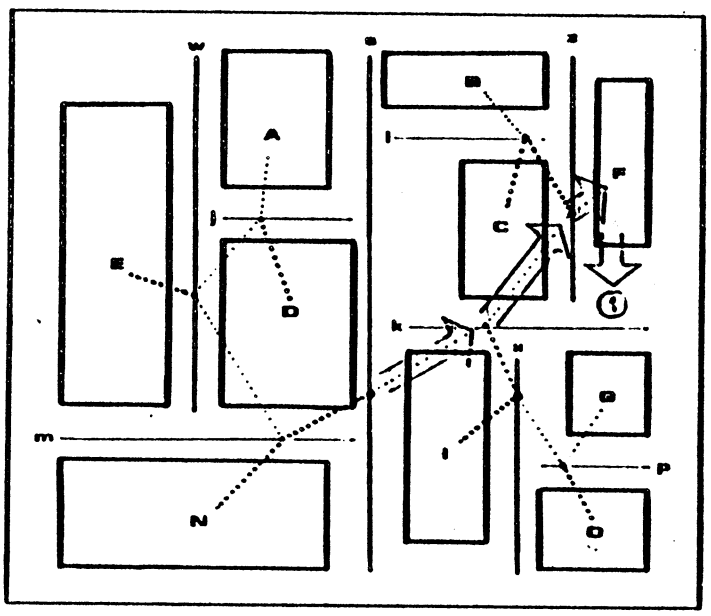


Fig IV-6. Un exemple de recherche d'un point.

Plus précisement, l'algorithme est décrit comme suit :

```
                        { i  est un interface ou bloc   }
                        { c  est l'ordonnée du point    }
                        { a  est l'abscisse du point    }
proc rech-point(i)
 cas type_de(i)
  'horizontale':si c<bas(i) alors rech-point(FI(i))
                sinonsi c>haut(i) alors rech-point(SU(FI(i))
                     sinon  " dans i"

  'verticale'  :si a<gauche(i) alors rech-point(FI(i))
                sinonsi a>droite(i) alors rech-point(SU(FI(i)))
                     sinon  " dans i "

  'bloc'       :si a<gauche(i) alors "a gauche de i"
                sinonsi a>droite(i) alors "a droite de i"
                     sinonsi c>haut(i) alors "en haut de i"
                          sinonsi c<bas(i) alors "en bas de i"
                               sinon  " dans i"
fincas.
```

4.2.1.4.2. Création d'un bloc.

Pour créer un bloc, il faut que l'espace désigné soit libre. Dans la représentation par interface, un espace est libre s'il appartient à la zone libre d'un bloc, dans le plan. Les étapes de création sont les suivantes :

1. Vérifier que la position du bloc créé ne traverse aucun interface ou bloc.
2. Si l'étape 1) n'est pas valide, l'espace n'est pas libre. Si elle l'est, il y a alors un bloc dont la zone libre peut contenir l'espace designé.
3. Créer un interface entre le bloc qui possède la zone libre et le bloc inséré.

La vérification dans l'étape 1) est réalisée par l'algorithme suivant :

```
                    { i est un interface ou bloc                    }
                    { b,d,g,h sont respectivement le côté           }
                    { bas, droite, gauche et haut du bloc créé}
proc verif-espace(i)
 cas type_de(i)
  'horizontale':si d≤droite(i) alors verif-espace(FI(i))
                sinonsi g≥gauche(i) alors verif-espace(SU(FI(i)))
                     sinon   "non libre"

  'vertical'   :si h≤haut(i) alors verif-espace(FI(i))
                sinonsi b≤bas(i) alors verif-espace(SU(FI(i)))
                     sinon  "non libre"

  'bloc'       :si b≥haut(i) alors "libre"
                sinonsi h≤bas(i) alors "libre"
                     sinonsi g≥droite(i) alors "libre"
                          sinonsi d≤gauche(i) alors "libre"
                               sinon   "non libre"
 fincas.
```

La complexité de cet algorithme est proportionnelle à la racine

carré du nombre d'interfaces dans le plan. Dans le pire cas, elle peut être égale au nombre total d'interfaces.

La figure ci-dessous montre un exemple de création d'un bloc (X).

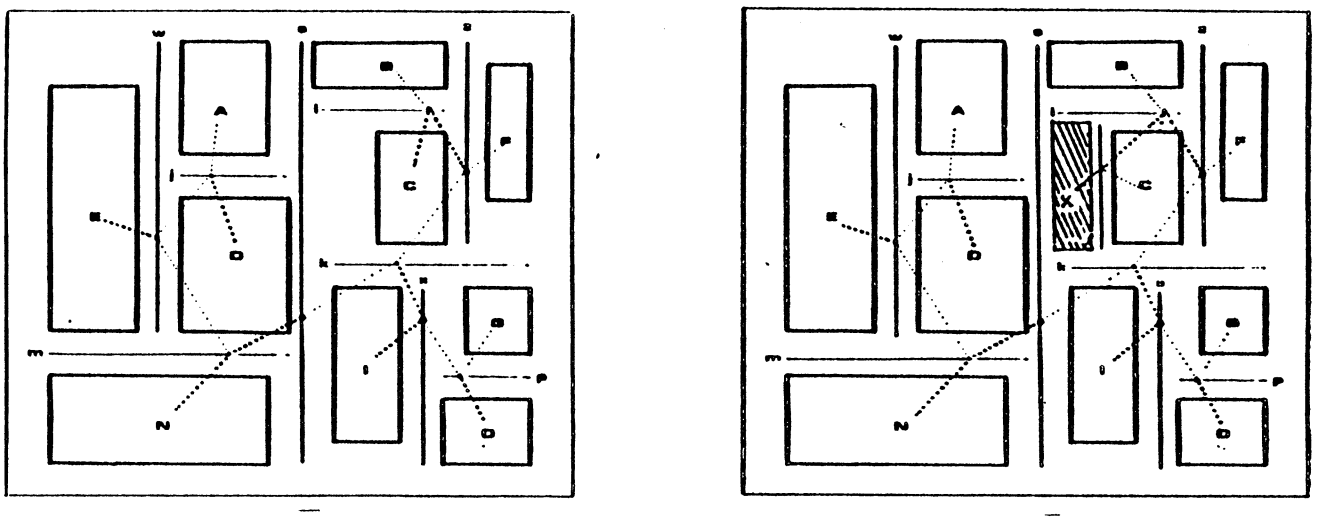


Fig IV-7. Exemple de création d'un bloc (X).
(a) état des liaisons avant la création
(b) état des liaisons après la création

4.2.1.4.3. Suppression d'un bloc.

Cette opération ne provoque pas d'incohérences topologiques. Lorsqu'un bloc est supprimé, l'interface pointé par un tel bloc, doit aussi être supprimée.

Le côut de suppression est constant, quelque soit le nombre des blocs dans le plan.

L'espace laissé par un bloc supprimé, peut être récupéré par utilisation de la propagation négative. A titre d'exemple, nous illustrons dans la figure ci-dessous, une suppression suivie d'une propagation négative.

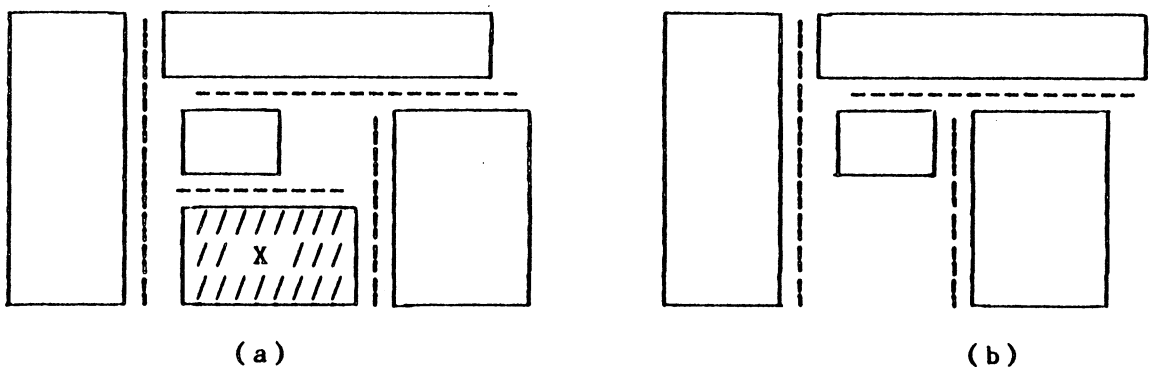


Fig IV-8. Exemple une suppression (bloc X)
suivi par une propagation negative.
(a) configuration avant la suppression
(b) configuration après la suppression

4.2.1.5. Mécanisme d'estimation des canaux de routage.

Dans la représentation par interface, l'information des canaux est portée par les interfaces. La détermination de l'ensemble des canaux qui peuvent être alloués à une interconnexion, est analogue à la recherche d'un chemin reliant deux noeuds dans l'arbre. Chaque élément du chemin correspond à une interface horizontale ou verticale.

Comme nous avons indiqué à la section 4.1.2, l'estimation des canaux de routage se fait en deux étapes : la détermination d'un trajet et l'affectation des canaux.

Détermination d'un trajet.
Soit i et j deux blocs à relier. CH(i) est un ensemble des interfaces de i à la racine, et CH(j) est un ensemble des interface de j à la racine.
A partir de CH(i) et CH(j) on peut déterminer le chemin de i à j qu'on appellera chemin structurel.
Le chemin topologique (cf. 4.1.2) peut être déterminé à partir du chemin structurel en supprimant les interfaces qui sont autour des blocs à relier, sauf la dernière.
Le parcours pour la suppression d'interfaces dans le chemin structurel est identique au parcours pour la recherche de la zone libre (cf. 4.2.2.2).

Exemple :

Soit D et G, dans la figure ci-dessous, deux blocs à relier. Le chemin structurel de D à G est une liste (D,j,w,m,s,k,x,p,G). Pour obtenir le chemin topologique, les interfaces à supprimer sont p et x pour le bloc G et j, w et m pour le bloc D. Le chemin final ne contient donc que D, s, k et G.

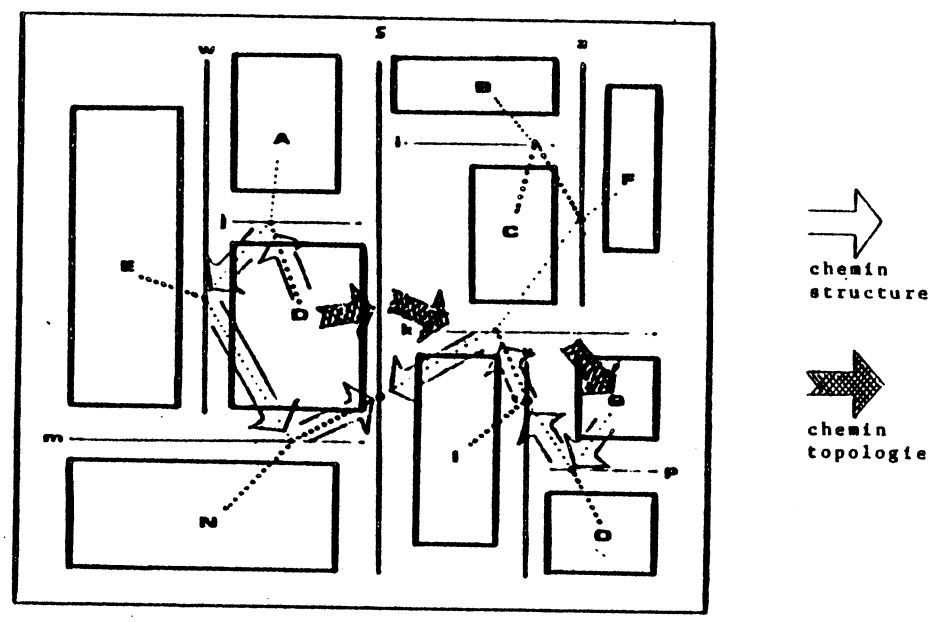


Fig IV-9. Exemple de détermination d'un trajet.

Affectation.

Sur chaque interface dans le trajet obtenu, on affecte une valeur poids (cf 4.1.2) selon le type d'interface.

Affectation de transparence. On affecte sur le trajet la transparence d'un bloc, si les conditions suivantes sont satisfaites :

- le trajet obtenu contient deux interfaces paralléles
- il existe entre ces deux interfaces un bloc qui accepte des interconnexion, i.e. ce bloc possède suffisamment de transparence dans la direction appropriée
- l'utilisation d'une transparence donne un chemin topologique plus court que la non utilisation de cette transparence.

Si c'est le cas on affecte le fragment des interconnexions à la transparence.

4.2.1.6. Problème ouvert.

La problème dans la représentation par interface est qu'il n'est pas possible d'avoir une configuration dit "pinwhell" [HAS-82] comme le montre la figure ci-dessous.

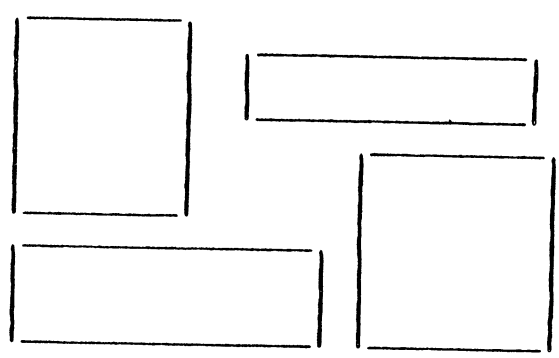


Fig IV-10. Un exemple de configuration pinwhell

La solution proposée dans [MAT-84] consiste à partager l'un des quatres blocs formant pinwhell en deux morceaux et établir une liaison spéciale pour ces morceaux.

4.2.2. REPRESENTATION PAR SEPARATEUR.

4.2.2.1. Introduction.

Le deuxième type de représentation interne que nous avons étudié est basé sur le découpage de la surface libre à l'aide de "ciseaux" horizontaux et verticaux.
Ce découpage peut être considéré comme une généralisation du découpage à l'aide des guillotines. Dans ce dernier cas, les lignes de coupure doivent toujours diviser entièrement le plan, ou sous-plan.

4.2.2.1.1. Principe.

On partage le plan à l'aide de lignes de coupure passant entre les blocs. Ces coupures s'arrêtent lorsqu'un bloc est rencontré ou lorsqu'elles sortent du rectangle d'encombrement des blocs.

La figure ci-dessous montre un exemple de plan coupé suivant cette méthode :

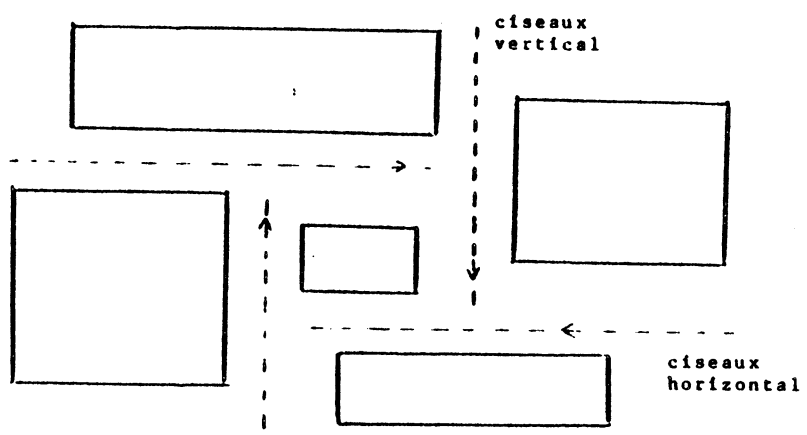


Fig IV-11. Exemple de coupure d'un plan à l'aide de ciseaux verticaux et horizontaux.

La trace des ciseaux joue le rôle d'objet invisible. Elle est appelée **"séparateur"**. Un séparateur représente symboliquement l'espace vide entre des blocs et porte la relation de voisinage entre ces blocs.

Lorsqu'un bloc bouge, il pousse un séparateur, qui à son tour pousse d'autres blocs. L'opération se répète, jusqu'au dernier séparateur déplacé.

Si dans la représentation par interface, les organisations fonctionnelle et topologique sont représentées de façon homogène, en une seule hiérarchie. Il n'est pas de même pour la représentation par séparateurs. Il est nécessaire de gérer en même temps une structure hiérarchique et une structure de graphe interne à la hiérarchie.

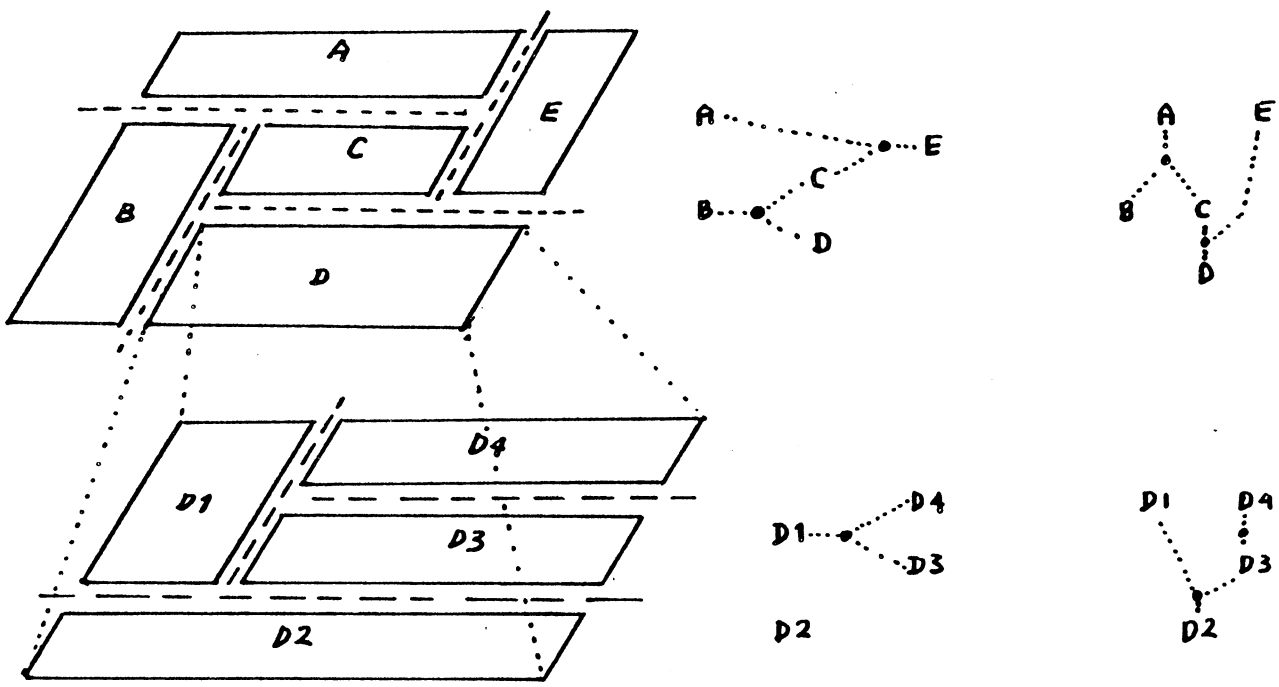


Fig IV-12. Graphes liés à chaque niveau.

4.2.2.1.2. Terminologie.

Chaque bloc est entouré par quatre séparateurs. Les blocs et les séparateurs sont liés de la manière suivante :

- Chaque séparateur vertical pointe sur l'ensemble des blocs qui se trouvent à sa droite et à sa gauche. De même, chaque séparateur horizontal, pointe sur l'ensemble des blocs qui se trouvent au dessous et au dessus du séparateur.
- Chaque bloc pointe sur les séparateurs horizontaux et verticaux qui l'entourent.

Cette liaison est décrite comme le montre la figure ci-dessous.

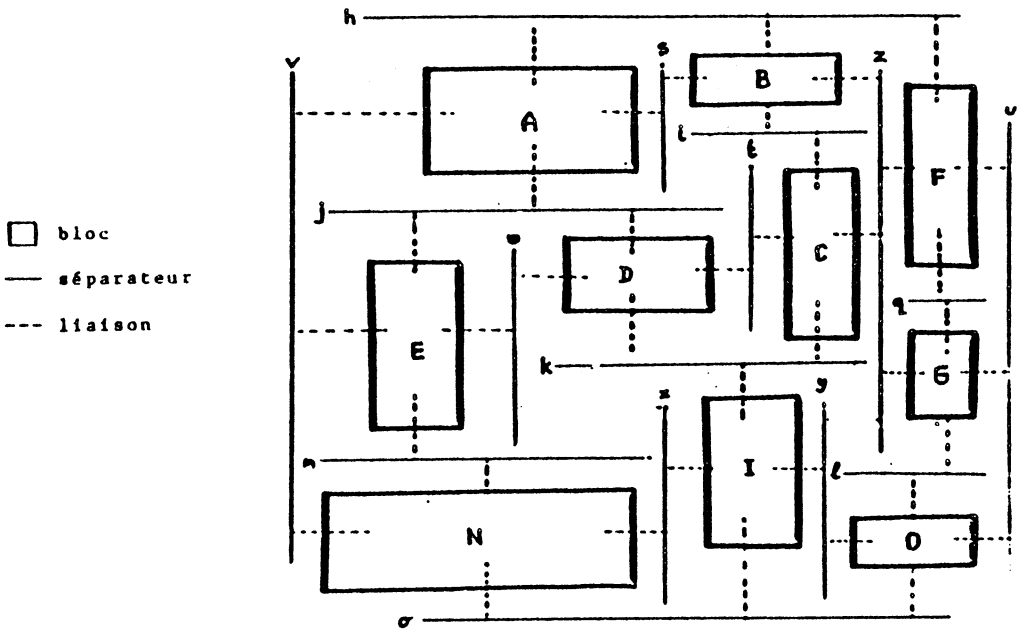


Fig IV-13. Liaisons entre blocs et séparateurs.

Ces liaisons forment un graphe planaire comme le montre la figure ci-dessous.

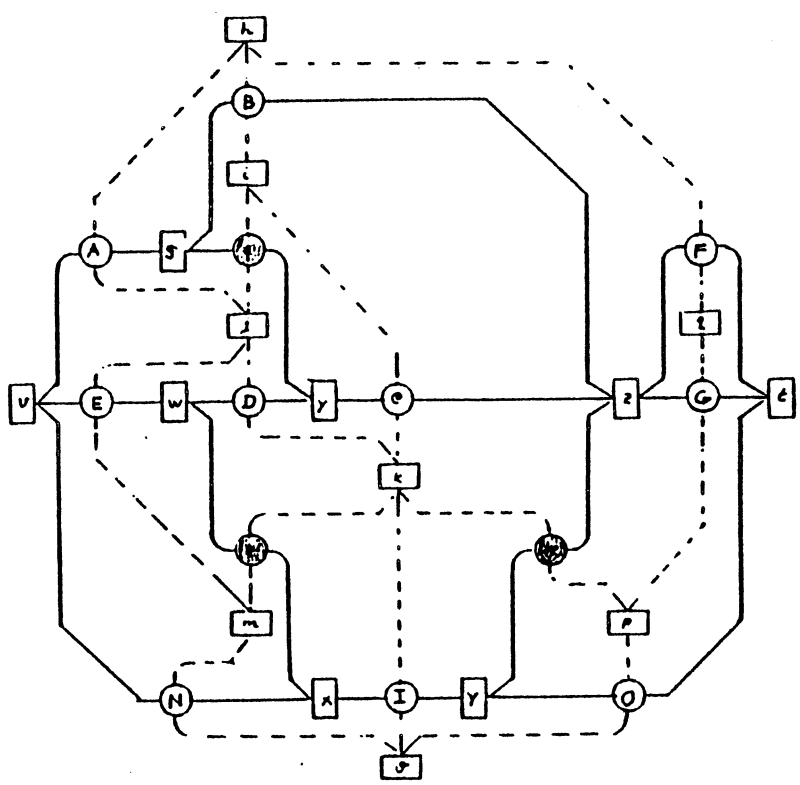


Fig IV-14. Graphe correspondant à la figure IV-13.

Ce graphe est décomposé en deux graphes :

- le graphe NORD-SUD (-------)
- le graphe EST-OUEST (_____)

Les terms ci-dessous seront utilisés afin d'alléger l'écriture.

SN(b) : représente le séparateur nord du bloc b.
SE(b) : représente le séparateur est du bloc b.
SS(b) : représente le séparateur sud du bloc b.
SO(b) : représente le séparateur ouest du bloc b.

BN(h) : représente les blocs au nord du séparateur h.
BS(h) : représente les blocs au sud du séparateur h.
BO(v) : représente les blocs à l'ouest du séparateur v.
BE(v) : représente les blocs à l'est du séparateur v.

4.2.2.2. Relation de voisinage.

4.2.2.2.1. Détermination des voisins.

Chaque séparateur est au même titre que les blocs, un noeud dans les différents graphes. La recherche des voisins d'un bloc dans une direction donnée se fait en parcourant le graphe.

Voisins nord d'un bloc.
Ces sont l'ensemble des blocs noeuds du graphe NORD-SUD qui se situent juste au nord du noeud séparateur qui se trouve au nord du noeud bloc en question. Ils sont formalisés de la manière suivante.

$$VN(b) = \{ lb \mid lb \in BN(h),\ h=SN(b) \}$$

Voisins ouest : $VO(b) = \{ lb \mid lb \in BO(v),\ v=SO(b) \}$

Voisins sud : $VS(b) = \{ lb \mid lb \in BS(h),\ h=SS(b) \}$

Voisin est : $VE(b) = \{ lb \mid lb \in BE(v),\ v=SE(b) \}$

4.2.2.2.2. Détermination de la zone libre autour d'un bloc.

Les limites de la zone libre autour d'un bloc sont déterminées à partir des positions des voisins d'un tel bloc, de la manière suivante :

Limite nord : ZN(b) = minbas (VN(b)).
Limite ouest : ZO(b) = maxdroite (VO(b)).
Limite sud : ZS(b) = maxhaut (VS(b)).
Limite est : ZE(b) = mingauche (VE(b)).

où minbas est une fonction qui donne le bord inférieur du voisin nord situé le plus près. maxdroite, maxhaut et mingauche ont les significations analogues au sud, à l'est et à l'ouest.
A titre d'exemple, la zone libre autour d'un bloc est montrée par la figure ci-dessous.

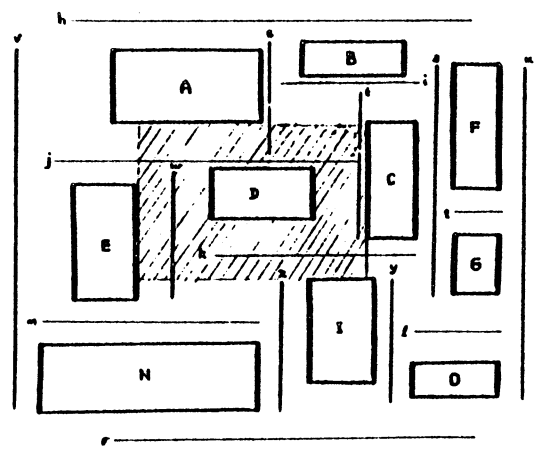


Fig IV-15. Exemple de la zone libre autour d'un bloc (hachurée) dans la représentation par séparateur.

4.2.2.3. Mécanisme de propagation.

En utilisant la relation de voisinage décrite précédement, le mécanisme de propagation est récursivement réalisé par l'algorithme ci-dessous.

Propagation positive dans la direction-est.

1. Déterminer les voisins-est du bloc modifié (déplacé) à l'aide de la recherche des voisins (cf 4.2.2.1).

2. Pour chaque bloc x voisin,
   - noter le déplacement nécessaire du bloc x pour le ramener au bord-est du bloc modifié (déplacé).
   - si la nouvelle position du bloc x dépasse la limite-est de sa zone libre (cf 4.2.2.2), appliquer l'étape 1 au bloc x.
3. Lorsqu'il n'y a plus des blocs à déplacer, mettre à jour la position des blocs.

La propagation positive dans l'autre direction est effectué de la même façon. La complexité de cet algorithme est proportionnelle au nombre des séparateurs visités et de blocs déplacés.

Propagation négative dans la direction-nord.

1. Déterminer les voisins-nord du bloc modifié (déplacé).
2. Pour chaque bloc x voisin
   - déterminer la limite sud de la zone libre du bloc x.
   - si cette limite est plus au sud que le côte sud du bloc x, alors déplacer le bloc x à cette limite et puis appliquer récursivement l'étape 1 au bloc x.

Cette propagation est deux fois plus complexe que la propagation positive.

Les figures ci-dessous montrent à titre d'exemple la propagation causée par la modification d'un bloc (X).

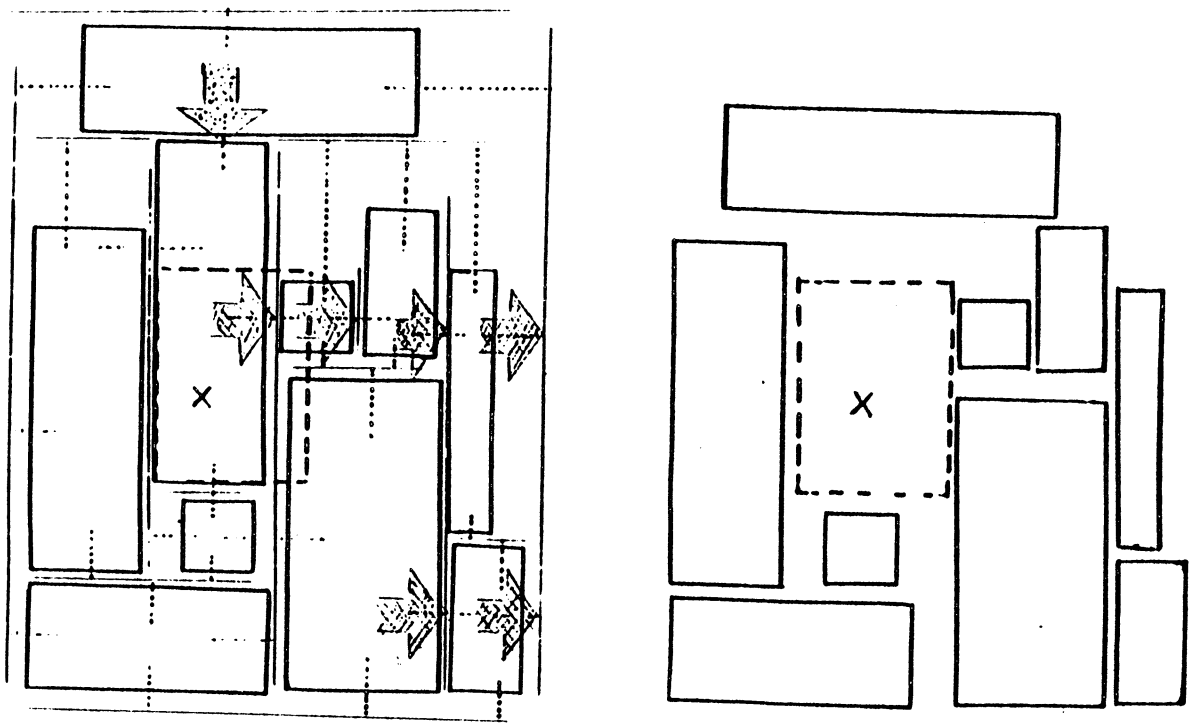


Fig IV-16 . Exemple de propagation.

4.2.2.4. Mécanismes de base.

4.2.2.4.1. Recherche d'un point.

La recherche d'un point a pour but de savoir si le point est dans un bloc. Elle est utilisée, par exemple, pour sélectionner un bloc.

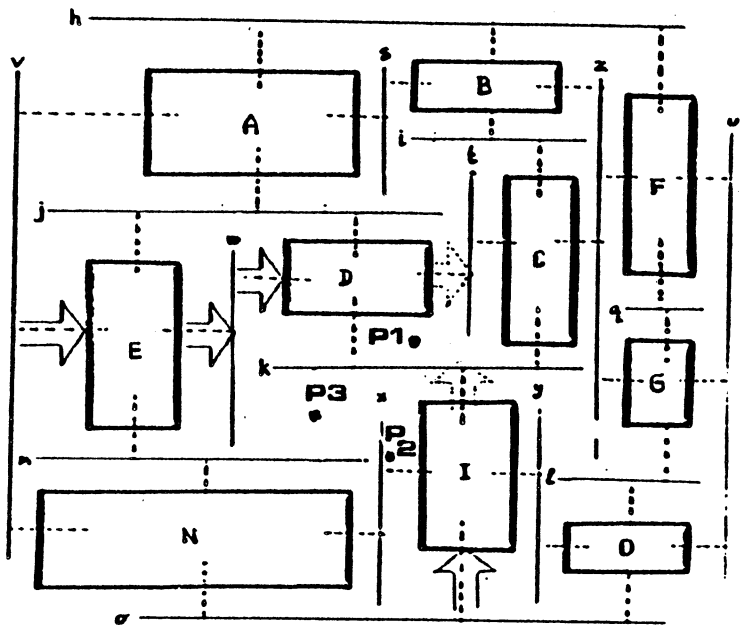


Fig IV-17. Exemple de recherche d'un point.

Cette recherche est basée sur un parcours dans le graphe NORD-SUD ou EST-OUEST en suivant les séparateurs. Par exemple, on commence le parcours sur le graphe EST-OUEST à partir du séparateur le plus gauche, l'algorithme est le suivant :

1. Prendre dans BE (cf. 4.2.2.1) un bloc dont les deux séparateurs horizontaux couvrent l'ordonnée du point recherché.
2. Lorsque l'abscisse du point est superieur à celui du séparateur-est du bloc choisi, passer à ce séparateur et exécuter 1.
   Dans le cas contraire, le point est dans le bloc ou dans sa zone libre.

Le coût de cet algorithme est proportionnel au nombre des blocs qui sont pointés par les séparateurs parcourus. Dans le pire des cas, le parcours passe sur tous les séparateurs. Cependant dans la mesure ou il est possible d'utiliser quatre points de départ différents, on peut choisir celui qui est le plus proche du point recherché. Le coût moyen est donc proportionnel à la racine carrée de (S+N)/2, où N est le nombre des blocs existants, S est le nombre total des séparateurs existants.

Dans la figure ci-dessus, la recherche du point P1 nécessite la

visite de deux séparateurs (i.e, v et w) et trois blocs (i.e, A, E et D), avec comme point de depart le séparateur v.
La recherche du point P2 est effectuée par une seule visite de séparateur(i.e, o) et une visite de bloc (i.e, I), avec comme point de depart le séparateur o.
Par contre pour la recherche du point P3, l'algorithme va visiter tous les blocs, sans trouver ce point, ceci quelque soit le séparateur de départ.
De plus, lorsque la modification, par exemple du bloc E, dépasse le séparateur x, il y a chevauchement entre les blocs E et I, sans que l'algorithme de propagation puisse le détecter.

La solution qui résoud ces deux problèmes, chevauchement et recherche d'un point, consiste à remplir une telle région par un bloc de taille nul, appelé **bloc bidon**. Le problème est alors de détecter l'existance d'une telle région et de lui affecter un bloc bidon. Ce problème est abordé dans la partie concernant la création des blocs (4.2.2.4.3).

4.2.2.4.2. Recherche d'espace.

La recherche d'espace a pour but de savoir si une partie d'un circuit est libre. Elle est utilisée, par exemple, afin de créer un nouveau bloc. Pour cette recherche on ne peut pas appliquer le parcours des graphes. La seule solution est de comparer un espace donnée à chaque bloc.

4.2.2.4.3. Création d'un bloc.

La création d'un bloc se fait en deux étapes. Premièrement il faut s'assurer que l'endroit où le bloc sera placé, est libre. Pour cela on utilise la recherche d'espace.

Deuxièmement il faut trouver et/ou créer éventuellement un ou des séparateurs, afin :

- d'entourer le bloc créé
- d'établit des liaisons.

Les futurs séparateurs entourant le bloc créé sont cherchés en parcourant les graphes NORD-SUD et OUEST-EST de façon à ce qu'elles

puissent former une plus petite région qui couvre la surface du bloc créé. On appele ces séparateurs des **"séparateurs couvrants"**.

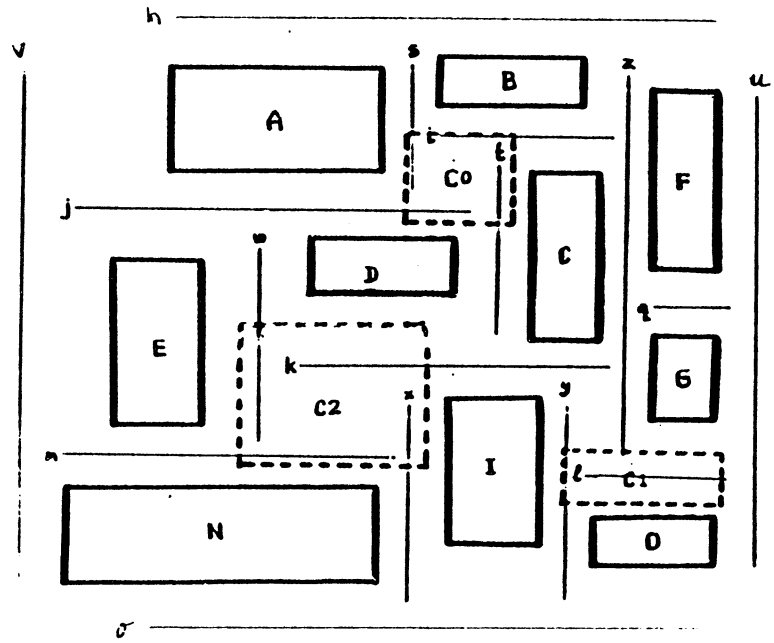
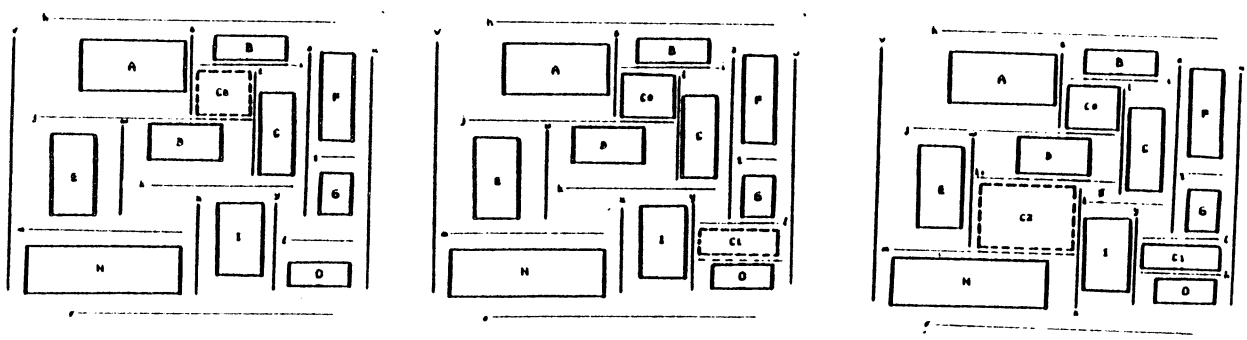


Fig IV-18. Exemples de création de blocs (C0, C1 et C2).
Les séparateur couvrants du bloc C2 sont m,w,j,t et x.
Ceux du bloc C0 sont j,s,i et t.
Ceux du bloc C1 sont o,y,l et u.



(a) (b) (c)

Fig IV-19. Etat des liaisons après la création des blocs C0 (a), C1 (b) et C2 (c).

Lors de mise à jour des liaison, deux cas peuvent se présenter.

Dans le premier cas, le nombre de séparateurs couvrants est égal à quatre. Par exemple (voir fig. IV-18), le cas de création du bloc C1 et C0. La mise à jour est comme suit :

1. Chercher le bloc qui est pointé simultanément par les séparateurs couvrants.
2. Créer un séparateur entre ce bloc et le bloc créé. Ensuite établir les liaisons.
3. établir des liaisons entre le bloc créé et les trois autres séparateurs couvrants.

Au niveau des séparateurs, la création des blocs C1 et C0 n'a pas les mêmes effets, car pour C0, le propriétaire des 4 séparateurs est un bloc bidon. Dans ce cas, aucun séparateur est crée. On remplace simplement le bloc bidon par C0.

Dans le deuxième cas, le nombre de séparateurs couvrants est supérieur à quatre. Cela veut dire qu'il y a un ou plusieurs séparateurs qui sont traversés le bloc créé. Dans la figure IV-18, c'est le cas de création du bloc C2, et le séparateur traversant est k. Ce séparateur doit être éclaté. L'algorithme de mise à jour des liaisons pour ce cas est assez compliquée. Il peut être décrit de la manière suivante :

1. Si entre un séparateur couvrant et le bloc créé il existe un autre bloc, créer un séparateur entre ce bloc et le bloc créé, et faire les liaisons.
2. Sinon créer les liaisons entre ce séparateur couvrant et le bloc créé.
3. Appliquer les étapes 1 et 2 aux autres séparateurs couvrants.

<u>remarques</u>.

- La création d'un nouveau séparateur dans une direction se fait en une seule fois.
- Lorsque une première liaison est faite sur un bloc créé la liaison suivante se fait toujours sur les séparateurs les plus proches de lui. Cependant à ce moment, une zone bidon se produit. Il est alors nécessaire de créer un bloc bidon.

Dans le premier cas la complexité d'algorithme de création est

proportionnelle à la somme du nombre des blocs et séparateur. Dans le deuxième cas, il faut ajouter le côut de mise à jour des séparateurs couvrants et éventuellement le côut des éclatements des séparateurs qui traversent le bloc créé.

4.2.2.4.4. Suppression d'un bloc.

La suppression d'un bloc ne provoque pas d'incohérence de configuration. Lorsqu'un bloc est supprimé un ou des séparateurs entourant ce bloc doivent être enlevés. On dit qu'un séparateur est enlevable par rapport à un bloc si'il ne lie que ce bloc. Par exemple (voir la figure ci-dessous), le séparateur enlevable du bloc F est q, les séparateurs enlevables du bloc I sont x et y, les séparateurs enlevables du bloc E sont w et m. Une remarque importante dans la détermination d'un séparateur enlevable, est que le bloc bidon est considéré comme nul, mais la mise à jour des séparateurs doit en tenir compte.

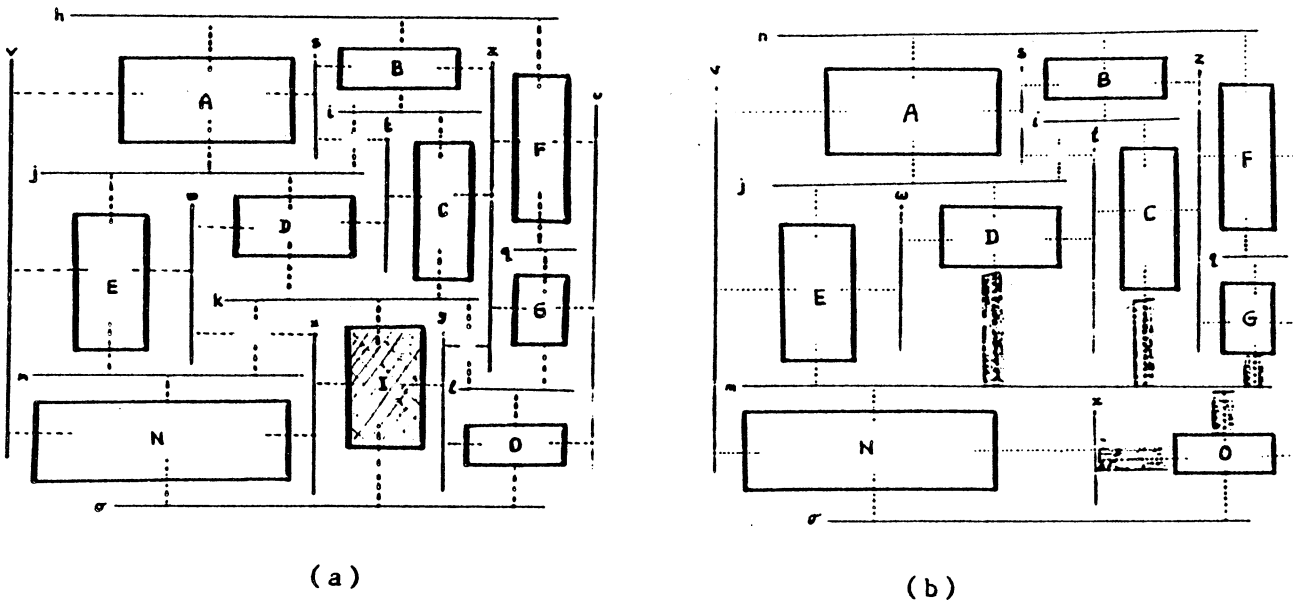


Fig IV-20. Exemple de suppression d'un bloc.
(a) état des liaisons avant la suppression
(b) état des liaisons après la suppression

Pour la mise à jour des séparateurs, lors de la suppression d'un bloc, trois cas peuvent se présenter :

- Lorsque le séparateur enlevable ne lie aucun bloc bidon, on l'enleve. (exemple, cas du bloc F)

- Lorsque le séparateur enlevable lie un bloc bidon, alors une fusion des séparateurs doit être effectuée. (exemple, cas des bloc E et I).
- Lorsque le séparateur enlevable n'existe pas (par exemple, cas du bloc D), aucun séparateur ne doit être enlevé. Le bloc supprimé devient un bloc bidon.

Dans le cas général, le coût de la suppression d'un bloc est constant. Mais dans un cas particulier, le deuxième, ce coût est proportionnel à celui d'une fusion de séparateurs.

4.2.2.5. Mécanisme d'estimation des canaux de routage.

Dans la représentation par séparateurs, l'information des canaux est disponible sous forme de séparateurs. La détermination de l'ensemble des canaux qui peuvent être alloués à un trajet d'interconnexions entre deux blocs est analogue à la recherche d'un point à partir d'un lieu donné.

Dans la recherche d'un point le graphe à parcourir est choisi librement, alors que dans la recherche d'un trajet ce n'est pas le cas. Aucun critère n'autorise la préference d'un graphe à l'autre. D'une façon heuristique on choisit le graphe à parcourir en se basant sur les positions relatives des blocs à relier. La relativité se traduit par la distance verticale et horizontale entre les deux blocs, comme le montre la figure ci-dessous.

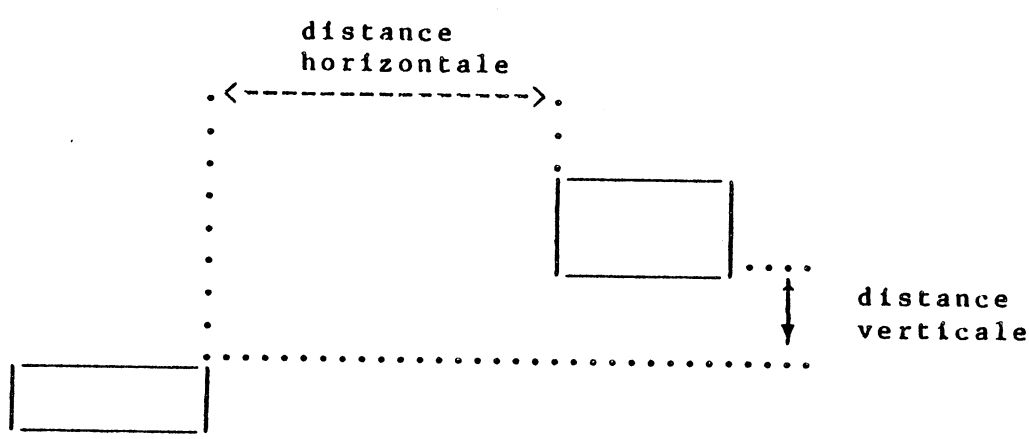


Fig IV-21. Position relative des blocs.

Comme nous l'avons indiqué à la section 4.1.2, l'estimation des canaux se fait en deux étapes : recherche d'un trajet et affectation d'un valeur poids.

Recherche d'un trajet.

En utilisant la méthode heuristique la recherche d'un trajet est la suivante :

1. Déterminer la position relative des deux blocs à relier.
2. Choisir le graphe à parcourir en fonction de cette position. Prendre le graphe EST-OUEST lorsque la distance horizontale est la plus longue, et le graphe NORD-SUD lorsque la distance verticale est la plus longue.
3. En partant de l'un des deux blocs, appliquer la recherche d'un point et noter les noeuds visités dans une liste appelée L.

Affectation.

Dans cette étape, on attribue une valeur poids (cf 4.1.2) à chaque séparateur contenu dans L. Et pour chaque bloc contenu dans L deux cas possibles existent:

- le bloc accepte des interconnexions, i.e. il possède suffisamment de transparence dans la direction approprié. Dans ce cas on affecte ces interconnexions à la transparence.
- le bloc ne possède pas de transparence. Dans ce cas la valeur poids doit être attribuée à l'un des deux séparateurs pointés par un tel bloc et qui n'appartient pas au graphe parcouru.

La figure ci-dessous montre un exemple de la recherche d'un trajet (de E à F).

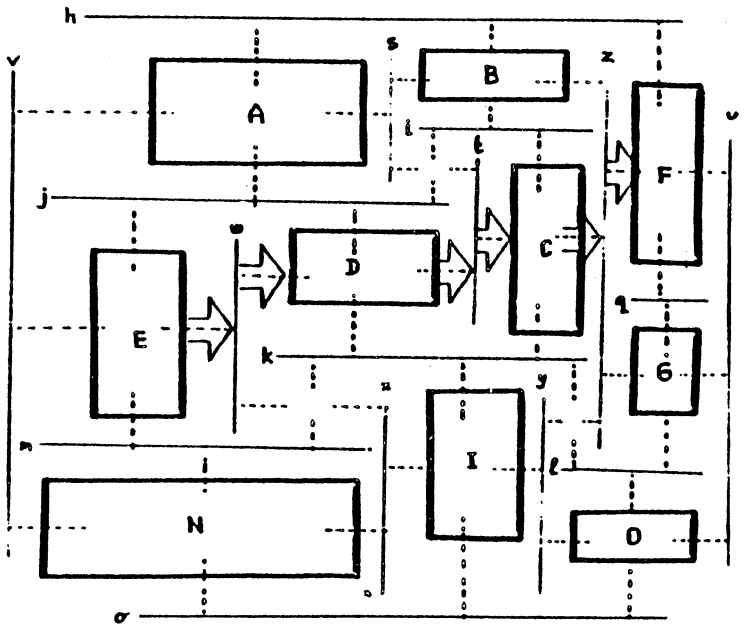


Fig IV-22. Exemple de recherche d'un trajet.

4.2.3. REPRESENTATION PAR TUILE.

4.2.3.1. Introduction.

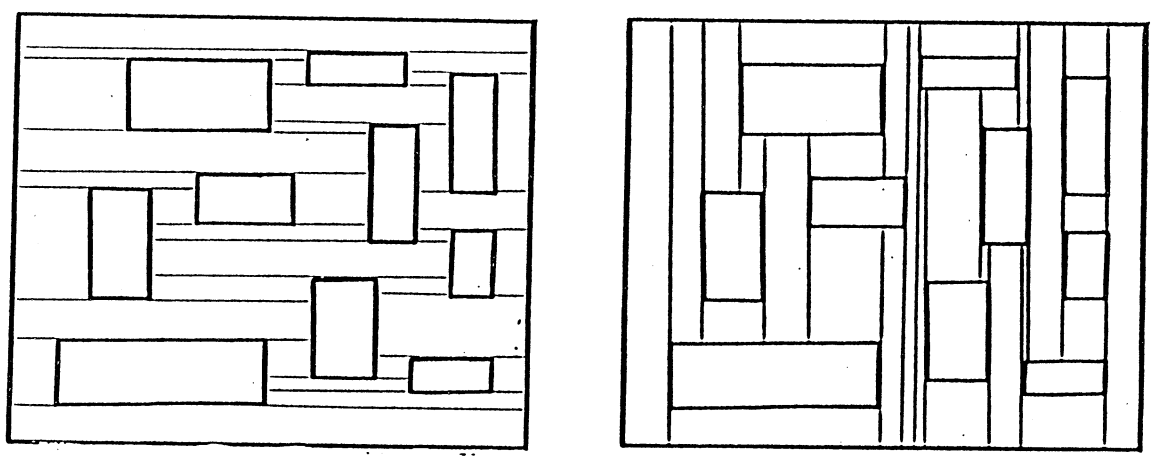
Le troisième type de représentation que nous avons étudié est basé sur le découpage de la surface libre à l'aide de lignes parallèles. On trouve ce type de découpage, dans [LEB-83, OUS-84].

4.2.3.1.1. Principe.

On partage le plan à l'aide des lignes de coupure passant sur le bord d'un bloc. Les lignes sont prolongées jusqu'à ce que l'un des cas suivants apparaisse :

- rencontrer un autre bloc
- sortir de la zone d'encombrement des blocs.

La figure ci-dessous montre un plan découpé suivant cette méthode :



(a) (b)

Fig IV-23. Exemple de coupure du plan par l'allongement deux bords parallèles.

La figure IV-23 est obtenue en prolongant les bords de blocs qui sont parallèles à l'axe des X (cas a), à l'axe des Y (cas b). Pour ce qui suit nous utilisons un seul modèle, i.e. celui de coupure horizontale suivant X.

Comme nous le voyons dans la figure ci-dessus, la surface libre est divisée en un ensemble de blocs vides. Dans cette représentation,

ces blocs vides jouent le rôle d'objets invisibles. La notion de tuile ("tail" dans [OUS-84]) est dorenavant utilisée, pour désigner un bloc vide.

4.2.3.1.2. Terminologie

Dans cette représentation, chaque bloc est entouré au moins par 4 tuiles. Plusieurs façons de lier des blocs et des tuiles. La liaison la plus intéressante, est celle du système MAGIC [OUS-84]. Elle est montree dans la figure ci-dessous.

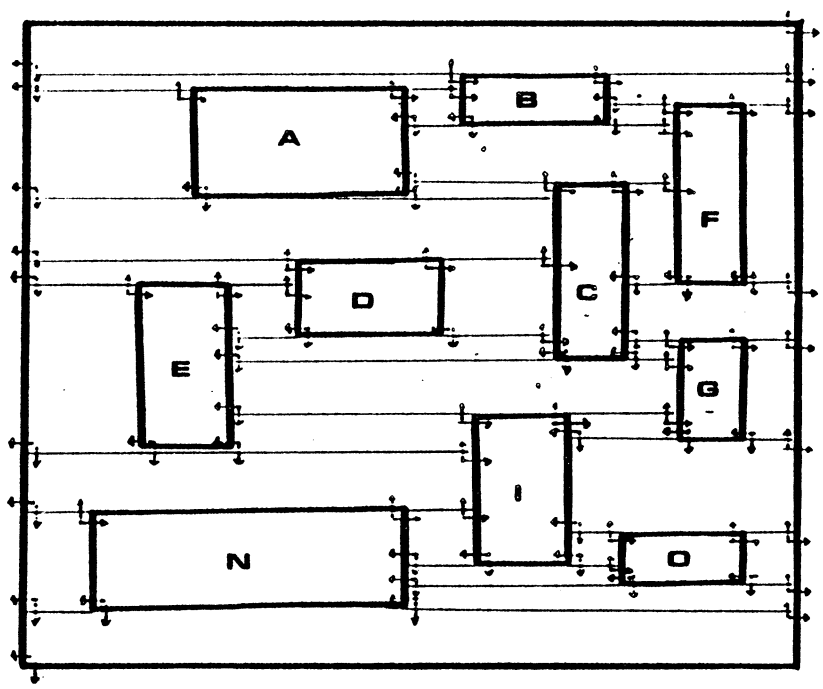


Fig IV-24 . Exemple de liaison entre blocs et tuiles.

La liaison est réalisée comme suit :
Chaque rectangle pointe sur les quatre rectangle situés , au nord, à l'est de son coin supérieur droit, au sud et à l'ouest de son coin inférieur gauche. Ces pointeurs sont appelés respectivement NORD, SUD, OUEST et EST.

Les liaisons obtenues correspondent à quatre graphes, comme le montre la figure ci-dessous.

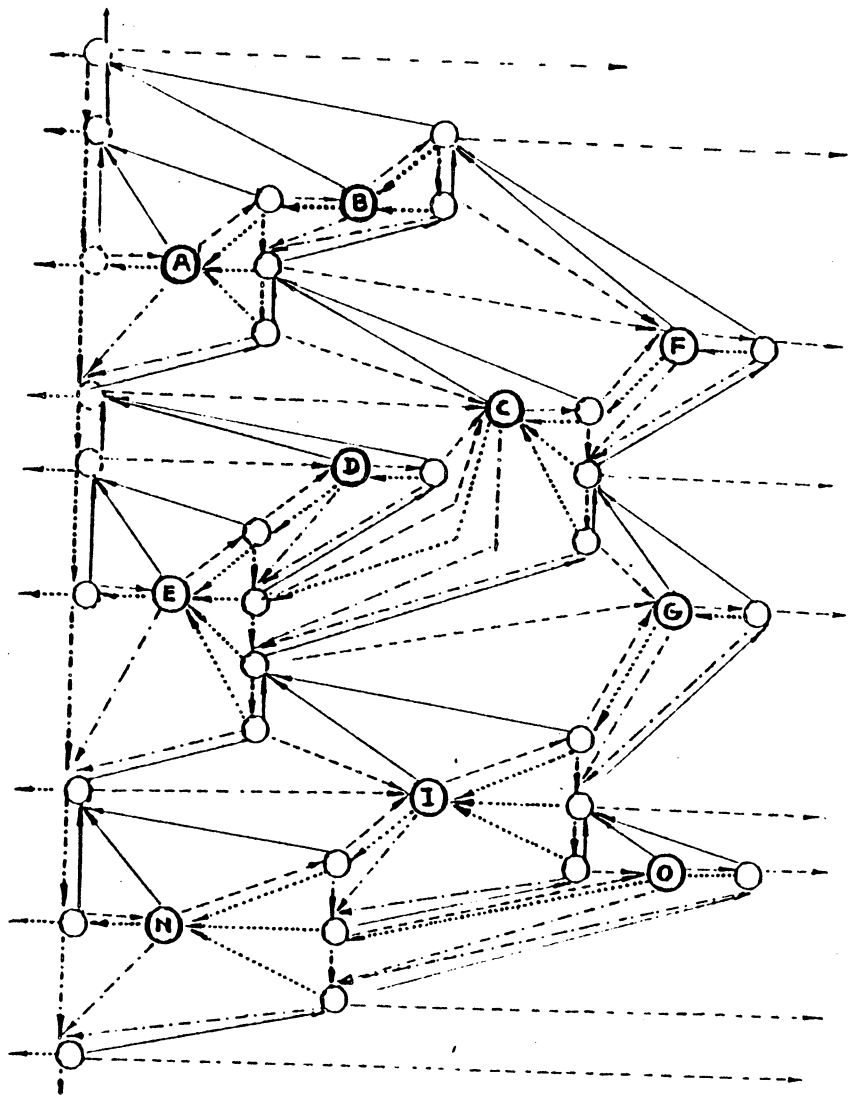


Fig IV-25. Graphes correspondants à la figure IV-23.

Chaque graphe est répéré par :

graphe NORD (———),
graphe EST (- - - -),
graphe SUD (—.—.—.—) et
graphe OUEST(.....).

Les blocs doivent toujours être séparés par une tuile, aussi petite soit elle. Sans quoi la translation d'un bloc touchant un autre bloc provoque des incohérence de représentation interne.
A titre d'exemple, la figure ci-dessous montre cette situation.

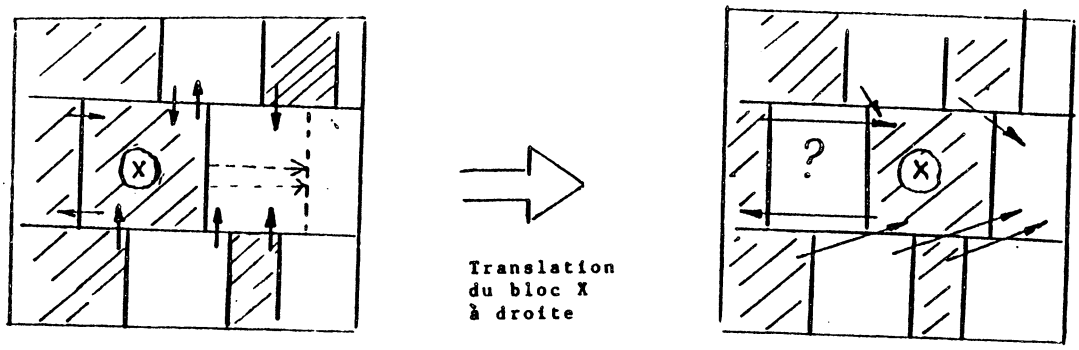


Fig IV-26.

En utilisant une tuile de taille nulle appelée **bidon**, lorsque deux blocs se touchent le problème d'incohérence disparait. Par l'utilisation des tuiles bidons la relation entre les blocs et les tuiles est exprimée comme suit :

NORD(tuile) --> tuile	NORD(bloc) --> tuile
SUD (tuile) --> tuile	SUD (bloc) --> tuile
EST (tuile) --> bloc	EST (bloc) --> tuile
OUEST(tuile) --> bloc	OUEST(bloc) --> tuile

où la notation "-->" se lit "est un(e)".

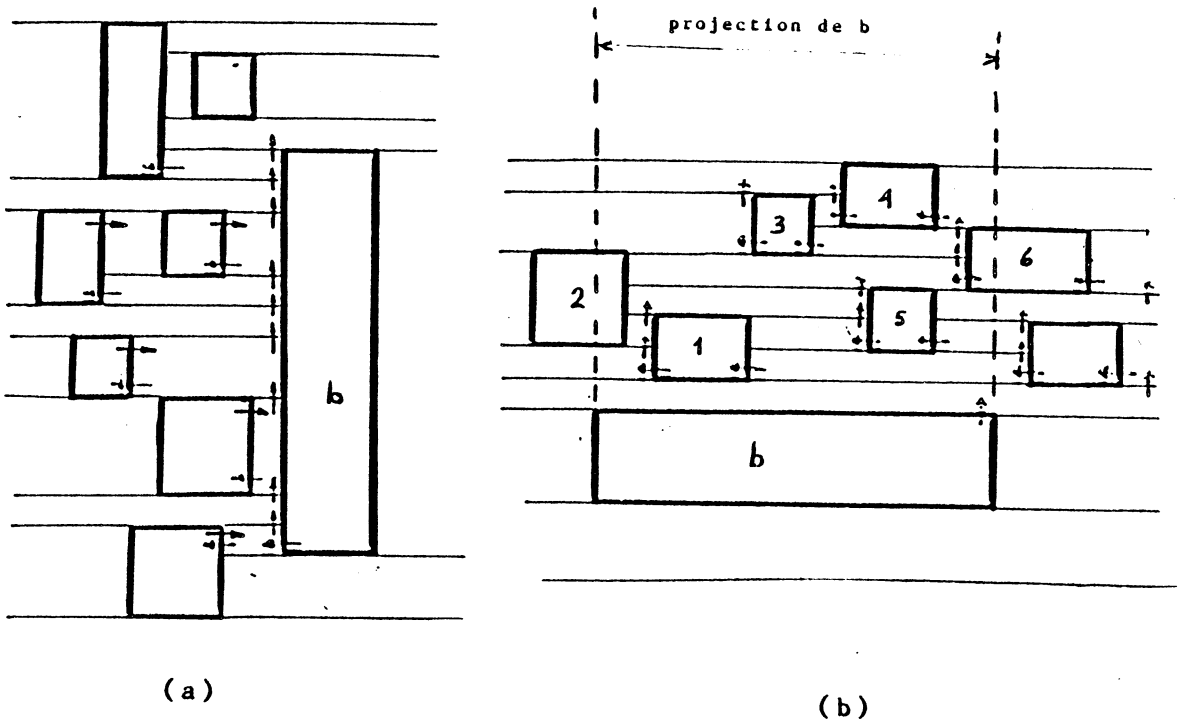
4.2.3.2. Relation de voisinage.

4.2.3.2.1. Recherche des voisins.

Avec ce type de représentation, la recherche des voisins d'un bloc n'est pas aussi immédiate qu'avec les représentation précédantes. Un algorithme de visibilité est nécessaire. Cet algorithme visite les tuiles voisines des blocs.

Voisins-ouest.

Pour déterminer les voisins ouest d'un bloc b, il faut d'abord rechercher l'ensemble des tuiles touchant b à gauche, puis regarder le bloc à gauche de chaque tuile dans cet ensemble.



(a) (b)

Fig IV-27. Exemple de recherche des voisins d'un bloc.
(a) recherche voisins-ouest
(b) recherche voisins-nord

L'algorithme ci-dessous réalise cette recherche (voir figure IV-27a) :

1. Mettre des tuiles à l'ouest du b, dans CVO(b), à l'aide des visites suivantes :
   - visiter la tuile OUEST(b), notée v .
   - visiter la tuile NORD(v), et refaire cette visite jusqu'à ce que EST(v) ne soit pas b .
2. Les blocs voisins-ouest du bloc b sont :

$$VO(b) = \{ o \mid o=OUEST(v) \text{ et } EST(o)=v \text{ et } v \in CVO(b) \}$$

La complexité de cet algorithme est une fonction linéaire de nombre de tuiles voisines.

Voisins-est.

Les voisins-est sont déterminés de la même façon que les voisins-ouest, mais les graphes utilisés sont EST, SUD et OUEST.

$$VE(b) = \{ e \mid e=EST(v) \text{ et } OUEST(e)=v \text{ et } v \in CVE(b) \}$$

Voisins-nord.

Les voisins nord d'un bloc b sont déterminées en deux étapes . D'abord il faut rechercher l'ensemble des blocs dans la projection (une région bornée par l'allongement deux bords parallèles) du bloc b vers le nord. Ensuite, il faut rechercher sur cet ensemble les blocs faisant face au bloc b.
La recherche utilise les graphes NORD et OUEST. Elle est réalisée par l'algorithme suivant (voir figure IV-27b):

I. Recherche des blocs dans la projection de b.
   1. Visiter la tuile NORD(b), notée v.
   2. Soit o un bloc OUEST(v).
      Si bloc o est à droite de b, passer v égal OUEST(o)
      et marquer cette tuile.
      Si bloc o est dans la projection de b, marquer le bloc o
      et passer v égal OUEST(o) et marquer cette tuile.
   3. Répéter l'étape 2 en prenant v égal NORD(tuile_marquée),
      jusqu'à ce que la tuile visitée soit nulle
      ou que le bloc visité couvre la zone de projection.

II. Les blocs voisins nord de b sont :

$$VN(b) = \{\, n \mid n \in \text{blocs\_marqués}, \text{ et il n'existe pas } x \in \text{blocs\_marqués} \text{ telque } bas(x) < bas(n) \text{ et } droite(x) > gauche(n) \text{ et } gauche(x) < droite(n) \,\}$$

La complexité de cet algorithme est proportionnelle à la somme du nombre des tuiles et au carré du nombre des blocs dans la projection.

Voisin sud.
Les voisins sud d'un bloc sont déterminés de la même façon que les voisins nord, mais les graphes utilisés sont ceux du SUD et de l'EST.

4.2.3.2.2. Détermination de la zone libre autour d'un bloc.

Cette détermination utilise l'algorithme de recherches des voisins décrit précédement.

Limite nord. C'est le bord inférieur du première bloc rencontré dans l'algorithme de recherche des voisins nord.

Limite sud. C'est le bord supérieur du première bloc rencontré dans l'algorithme de recherche des voisins sud.

Limite ouest. C'est le bord droit du bloc le plus près parmi les voisins ouest.

Limite est. C'est le bord gauche du bloc le plus près parmi les voisins est.

A titre d'exemple la zone libre autour d'un bloc est montrée dans la figure ci-dessous.

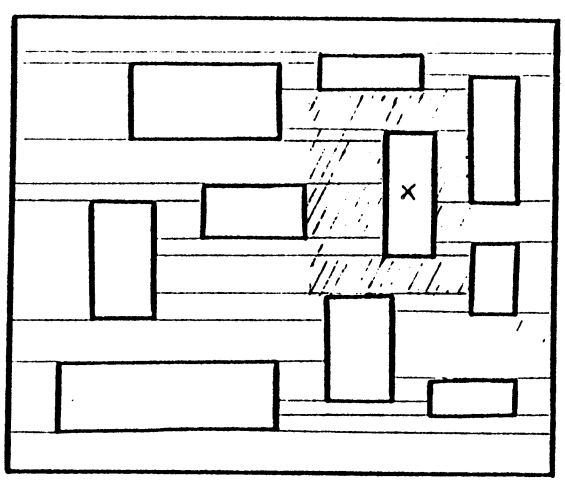


Fig IV-28. Exemple de zone libre autour d'un bloc dans la représentation par tuiles.

4.2.3.3. Mécanisme de propagation.

Avec la représentation par tuiles, le mécanisme de propagation dans les sens verticaux et horizontaux, n'est pas le même, car le déplacement dans le sens vertical nécessite le réarrangement des tuiles autour du bloc déplacé, afin de maintenir la cohérence de la

représentation interne. Dans le sens horizontal, ce réarrangement n'est pas nécessaire, à condition que les blocs se touchant soient séparés par une tuile bidon (cf. 4.2.3.1).

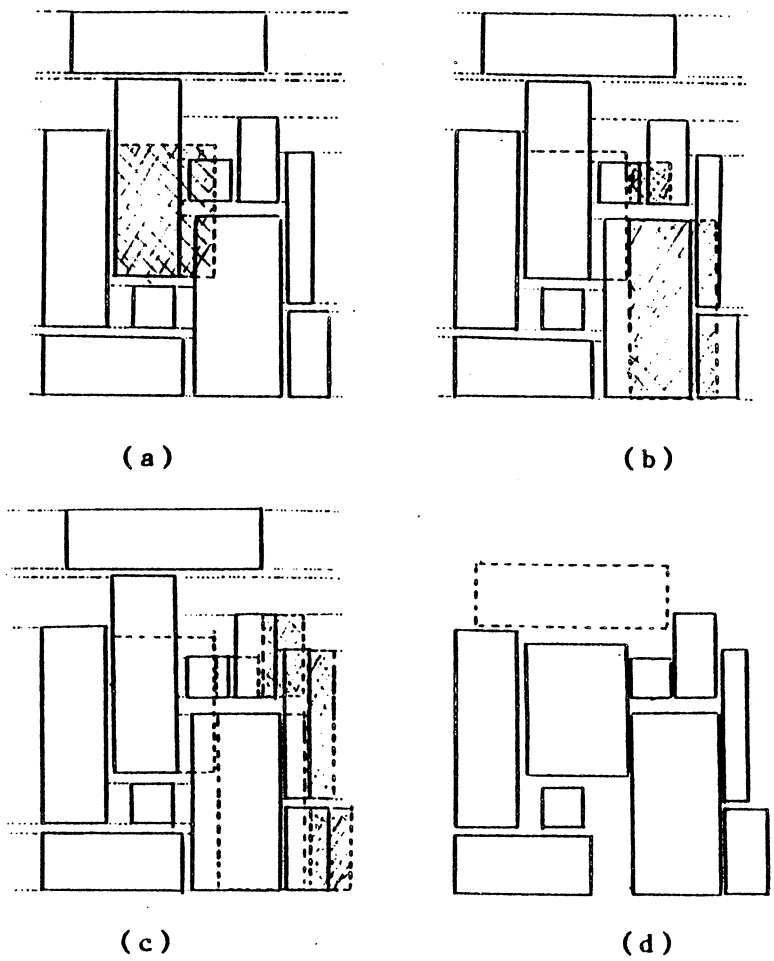


Fig IV-29. Exemple de propagation.

L'algorithme suivant décrit la propagation positive dans la direction-est.

1. Déterminer le côté droite, noté E, du bloc modifié (déplacé).
2. Déterminer la limite-est de la zone libre de tel bloc.
   Si celle-ci est dépassée par E, alors pour tous les voisins-est couverts par le bloc modifié (déplacé), marquer le déplacement de chacun pourqu'il soit à l'extérieur du bloc modifié.
   Recommencer l'étape 1 à chaque bloc déplacé.
3. Mettre à jour la position des blocs qui sont marqués.

Le côut de cet algorithme est proportionnel au nombre des blocs déplacés et des tuiles visitées.
De manière similiaire, on réalise la propagation dans la direction-ouest, nord et sud. Cependant l'étape 3 est changée par l'étape ci-dessous pour les directions nord et sud, car il faut réarranger des tuiles autour des blocs déplacés.

3. Supprimer les blocs déplacés de leur ancienne position et créer ceux-ci dans leur nouvelle position à l'aide de la liste D.

Le côut de la propagation dans la direction nord ou sud est proportionnel au nombre des blocs déplacés auquels il faut ajouter le côut de la suppression et la création de ces blocs.

4.2.3.4. Mécanismes de base.

4.2.3.4.1. Recherche d'un point.

Le parcours pour rechercher un point dans cette représentation peut utiliser, soit les graphes SUD et EST en partant de la tuile située le plus haut, soit les graphes NORD et EST en partant de la tuile située le plus bas.

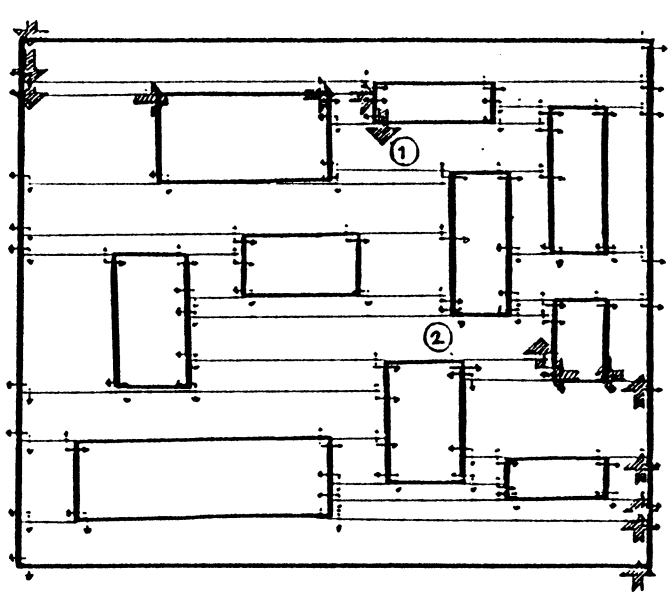


Fig IV-30. Exemple de recherche d'un point.

En prenant la tuile le plus haut comme point de départ l'algorithme est le suivant (voir figure ci-dessus) :

1. Passer au SUD, jusqu'à ce que l'ordonnée du point soit couverte par la tuile visitée. Si l'abscisse du point est couverte par cette tuile, alors le parcours est terminé, sinon passer à l'étape 2.

2. Passer à l'EST, jusqu'à ce que l'abscisse du point soit couverte par la tuile visitée. Si l'ordonnée du point est couverte par cette tuile, alors le parcours est terminé, sinon retour à l'étape 1.

Dans le cas où les formes des blocs sont presque uniforme, le point le plus long à trouver est celui qui est situé au centre du rectangle d'encombrement. Comme on peut utiliser deux points de départ différents, alors on choisit celui qui est le plus proche. Le temps moyen de parcours est proportionnel à (C+B)/4, où C est le nombre totale des tuiles et B le nombre total des blocs.
Il est intéressant de noter que dans cette représentation, une recherche peut partir de n'importe quel bloc ou tuile.

4.2.3.4.2. Recherche d'un espace.

Le parcours à suivre pour vérifier qu'un espace donné est libre, peut utiliser soit les graphes SUD et EST, soit les graphes NORD et OUEST.

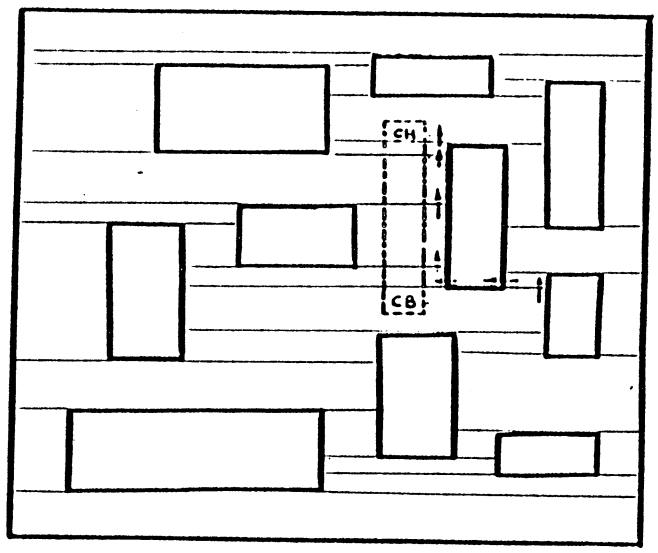


Fig IV-31. Exemple de recherche d'espace.

Prenons les graphes NORD et OUEST, la recherche est la suivante (voir la figure ci-dessus) :

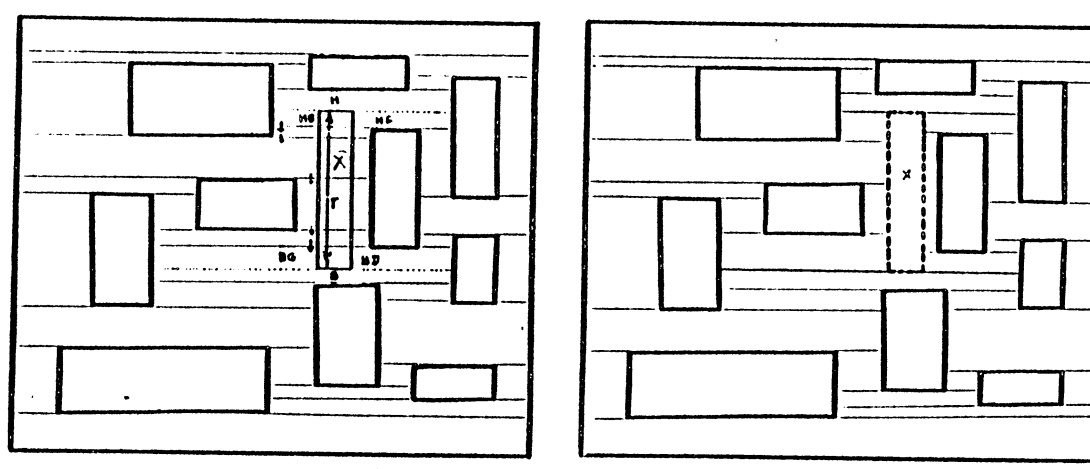
1. Trouver le rectangle, noté CB, qui contient le coin bas gauche de l'espace cherché. Pour cela appliquer l'algorithme de recherche d'un point. Tester si le rectangle est un bloc. Dans ce cas, la recherche est terminée, l'espace n'est pas libre. Dans le cas contraire, le rectangle est une tuile. Il faut alors vérifier que la largeur de l'espace libre cherché tient à l'intérieur de la tuile.
   Si ce n'est pas le cas, la recherche est terminée, l'espace n'est pas libre.
2. Appliquer 1) pour le coin haut droite de l'espace, le rectangle trouvé est noté CH.
3. Si CB est égal CH, alors l'espace est libre.
4. Rechercher des tuiles successives entre CB et CH, à partir de CB, avec la démarche suivante :
   a. Passer au NORD.
   b. Si la tuile trouvée correspond à CH, alors l'espace est libre.
   c. Si la largeur de l'espace tient à l'intérieur de la tuile, alors retour à l'etape a).
   d. Si l'espace est à la gauche de la tuile (il y a un bloc entre la tuile et l'espace), alors passer deux fois à l'OUEST et retour à l'étape b).

Sinon l'espace cherché n'est pas libre.

La complexité de l'algorithme pour rechercher un espace peut être mesurée par le nombre de tuiles traversées auquel s'ajouter la complexité de recherche d'un point.

4.2.3.4.3. Création d'un bloc.

La création d'un bloc se fait en deux étapes. Dans la première étape il faut assurer que l'endroit choisi est libre. Pour cela on applique l'algorithme de recherche d'espace. La deuxième étape est la mise à jour des structures. Les tuiles situées sous le bloc créé doivent être réammenagées afin de maintenir la cohérence de la représentation. Dans la rechercher de l'espace libre, la plus basse et plus haute tuile traversées sont notées CH et CB. Notons T_i les tuiles traversées (voir figure ci-dessous).



(a) (b)

Fig IV-32. Exemple de création d'un bloc (X).
(a) configuration des tuiles avant la création
(b) configuration des tuiles après la création

L'algorithme de mise à jour des tuiles est le suivant :

1. Casser la tuile CH en trois morceaux, notées HG, H et HD (voir figure ci-dessus). Mettre à jour les liaisons entre ces trois morceaux.
2. Même action pour CB que pour CH, avec BG, B et BD.
3. Ré-amenager les tuiles situées à gauche du bloc créé de la façon suivante :
   - En allant de HG à BG, à l'aide des tuiles T,

enlever la tuile à venir qui a le même OUEST que la tuile visitée.

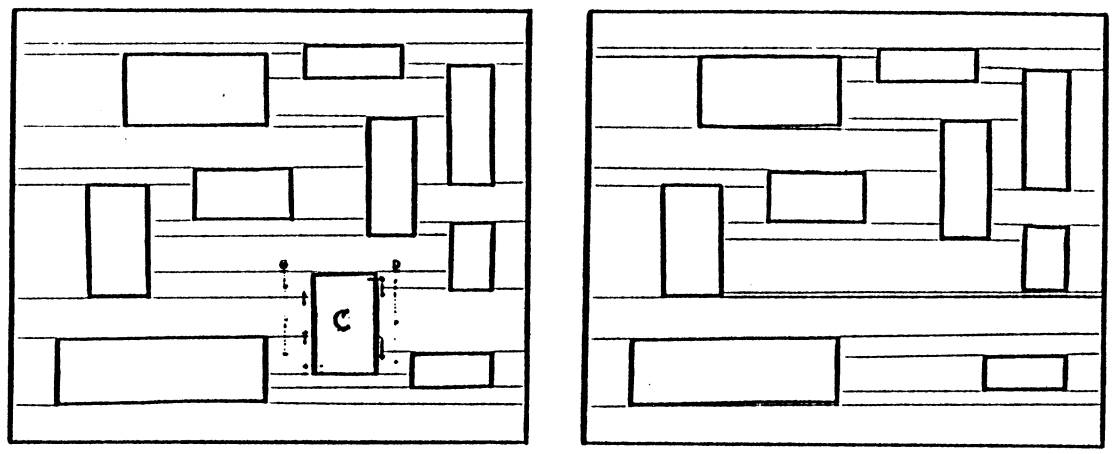
- Mettre à jour la liaison.

4. Même action qu'en 3, pour les tuiles à droite du bloc créé, en allant de BD à HD et EST comme bord comparé.

La complexité de l'algorithme de création d'un bloc est proportionnelle au double du nombre des tuiles traversées.

4.2.3.4.4. Suppression d'un bloc.

Dans la repésentation par tuile, la suppression d'un bloc provoque des incohérences sur les liaisons. Afin de maintenir cette cohérence, les tuiles autour d'un bloc supprimé doivent être réamenagées. Ce nouvel arrangement est appelé la fusion.



(a) (b)

Fig IV-33. Exemple de suppression d'un bloc (C).
(a) configuration des tuiles avant la suppression
(b) configuration des tuiles après la suppression

L'algorithme de mise à jour des tuiles est le suivant :

1. Trouver l'ensemble des tuiles à gauche du bloc supprimé, à l'aide de l'algorithme de recherche de tuiles voisines, notées G (voir figure IV-33a).
2. Même action pour les tuiles à droite, notées D.
3. Fusionner les tuiles G et D de la facon suivante:

a. Comparer la première tuile en haut de G ,notée g, et celle de D, notée d.
b. Si le coin bas gauche de g est plus bas que celui de d, allonger la tuile d à gauche et mettre à jour les liaisons. Passer à la tuile d suivante de la liste D et recommencer b).
c. Si le coin bas gauche de d est plus bas que celui de g, allonger la tuile g à droite, et mettre à jour les liaisons. Passer à la tuile g suivante de la liste G et recommencer b).
d. Si g et d ont leur coin bas gauche à la même hauteur, allonger l'une des deux tuiles et mettre à jour les liaison. Passer alors aux tuiles g et d suivantes et recommencer b.

Le cycle b), c) et d) se termine lorsque l'ensemble des tuiles appartenant à G et D a été traité.

La complexité de l'algorithme de suppression est proportionnelle au double du nombre des tuiles situées à la gauche et à la droite du bloc supprimé.

4.2.3.5. Mécanisme d'estimation des canaux de routage.

Dans la représentation par tuiles, l'information sur les canaux est disponible sous forme de tuiles. La détermination de l'ensemble des canaux qui peuvent être alloués à un trajet d'interconnexions, est analogue à la recherche d'un point à partir d'un lieu donné.

Les tuiles relevées au moment du parcours ne correspondent pas toujours du trajet le plus court. Il est nécessaire de faire une réduction sur le nombre des tuiles.

Nous décrivons un algorithme heuristique pour cette recherche, avec comme critère de réduction : les tuiles parcourues doivent être dans la région englobant les deux blocs à relier.

Détermination de trajet.

Soit i et j deux blocs à relier. L'algorithme de recherche est le

suivant:

1. Faire une fenêtre englobant les blocs i et j.
2. Partir de i en passant au SUD ou au NORD suivant la position relative de i par rapport à j.
3. Trouver une tuile à venir qui soit dans la fenêtre et qui touche la dernière tuile visitée. Ceci peut être effectué en visitant d'abord une tuile au SUD ou au NORD, puis passer 2 fois à l'EST ou à l'OUEST.
4. Répéter l'étape 3) jusqu'à ce que l'EST ou l'OUEST de la tuile visitée soit j.

Affectation.

Pour chaque tuile visitée, on affecte une valeur poids.

A titre d'exemple, la figure ci-dessous illustre une telle recherche (de A à O).

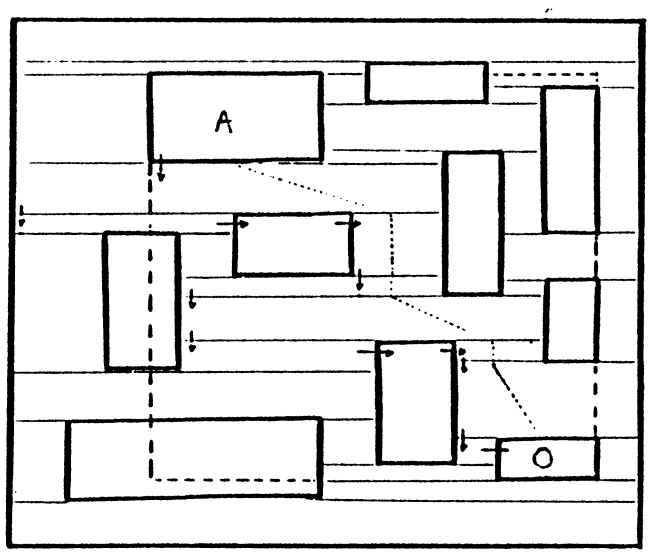


Fig IV-34. Recherche d'un trajet d'interconnexions.

4.2.4. REPRESENTATION PAR LISTE.

4.2.4.1. Introduction

Le quatrième type de représentation que nous avons étudié, est la représentation sans utilisation d'objet invisible. Les blocs sont liés dans une **liste simple.**
Lorsque la forme d'un bloc est modifiée, les autres blocs qui doivent être déplacés sont cherchés en utilisant la projection du côté d'un tel bloc dans la direction appropriée.

L'aspect intéressant de cette représentation, est qu'il n' y pas de traitement supplémentaire pour l'objet invisible. Mais toutes les opérations topologiques nécessitent un parcours total de la liste des blocs.

La figure ci-dessous donne un exemple de représentation par liste simple.

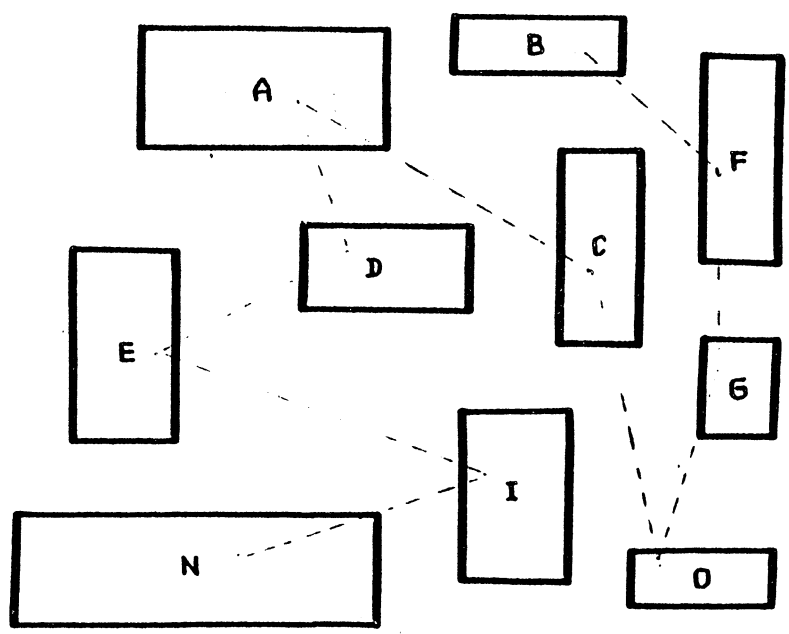


Fig IV-35. Représentation par une liste simple.

4.2.4.2. Relations de voisinage.

Les voisins d'un bloc dans une direction donnée sont déterminés à partir de la liste des blocs qui sont dans la projection d'un tel bloc.
On dit qu'un bloc p est dans la projection nord du bloc A si :

$$\begin{aligned} bas(p) &> haut(A) \text{ et} \\ droite(p) &> gauche(A) \text{ et} \\ droite(A) &> gauche(p) . \end{aligned}$$

Pour obtenir la liste des blocs dans la projection nord, il faut balayer tous les blocs. Les projections dans les autres directions sont déterminées de la même façon. On change simplement les côtès à comparer.

4.2.4.2.1. Détermination des voisins.

Soit P(b) l'ensemble des blocs dans la projection d'un bloc b. Un bloc v de P(b) est dit voisin du b, s'il n'y a pas d'autre bloc dans P(b) qui existe entre b et v.

Voisin nord.

$$\begin{aligned} VN(b) = \{\, v \mid\ & v \in P(b) \text{ et il n'existe pas } x \in P(b) \\ & \text{telque} \\ & \quad gauche(v) < droite(x) \text{ et} \\ & \quad gauche(x) < droite(v) \text{ et} \\ & \quad haut(v) < bas(x) \,\} \end{aligned}$$

La complexité de cette recherche est proportionnelle au carré du nombre des blocs dans P(b).
De manière analogue, on détermine les voisins dans les autres directions.

4.2.4.2.2. Détermination de la zone libre autour d'un bloc.

Les limites de la zone libre autour d'un bloc dans chaque direction sont déterminées à partir de l'algorithme de recherche des voisins.

Limite nord. C'est le plus petit bord inférieur des voisins nord.
Limite sud. C'est le bord supérieur le plus près des voisins sud.
Limite est. C'est le bord gauche le plus près des voisins-est.
Limite ouest. C'est le bord droite le plus près des voisins ouest.

4.2.4.3. Recherches topologiques.

4.2.4.3.1. Recherche d'un point.

Cette recherche est très simple. Mais elle nécessite un balayage total de la liste des blocs. Les conditions de recherche sont les suivantes :

```
haut(bloc)   > ordonnée(point) > bas(bloc)  et
droite(bloc) > abscisse(point) > gauche(bloc)
```

4.2.4.3.2. Recherche d'un espace.

Elle a pour but de savoir si un espace ne contient pas de bloc. La balayage pour cette recherche, utilise les conditions suivantes :

```
haut(bloc)   > bas(espace)    et
bas(bloc)    < haut(espace)   et
droite(bloc) > gauche(espace) et
gauche(bloc) < droite(espace)
```

Lorsque aucun bloc ne satisfait les conditions ci-dessus, alors l'espace est libre.

4.2.4.3.3. Enumération d'un espace.

L'énumération d'un espace est destinée à connaitre les blocs existants, dansun espace donné. Elle est donc exactement comme la projection, mais dans ce cas les conditions de recherche sont identiques à celles de la recherche d'un espace.

4.2.4.4. Mécanisme de propagation.

Propagation positive.

Pour permettre la propagation causée par la modification de la forme d'un bloc, il faut trouver les voisins dans la direction concernée. Pour cela, on applique d'abord l'énumération de l'espace, à partir de celle-ci on recherche les voisins. Il faut remarquer ici que l'utilisation de l'énumération d'espace au lieu de la projection, permet de diminuer d'un carré la complexité de la recherche des voisins.

Les voisins trouvés sont déplacés jusqu'à l'extérieur de l'espace énuméré. Cependant, il est probable qu'entre l'ancienne et la nouvelle position d'un bloc déplacé, existe un ou plusieurs blocs. Si c'est le cas, il y a alors un bloc sauté.

Pour éviter cela, avant de déplacer un bloc, il faut faire recursivement l'énumération sur l'espace qui couvre l'ancienne et la nouvelle position d'un tel bloc, et déterminer les voisins à "déplacer". Ainsi de suite, jusqu'à ce que les blocs à déplacer ne dépassent pas leur zone libre.

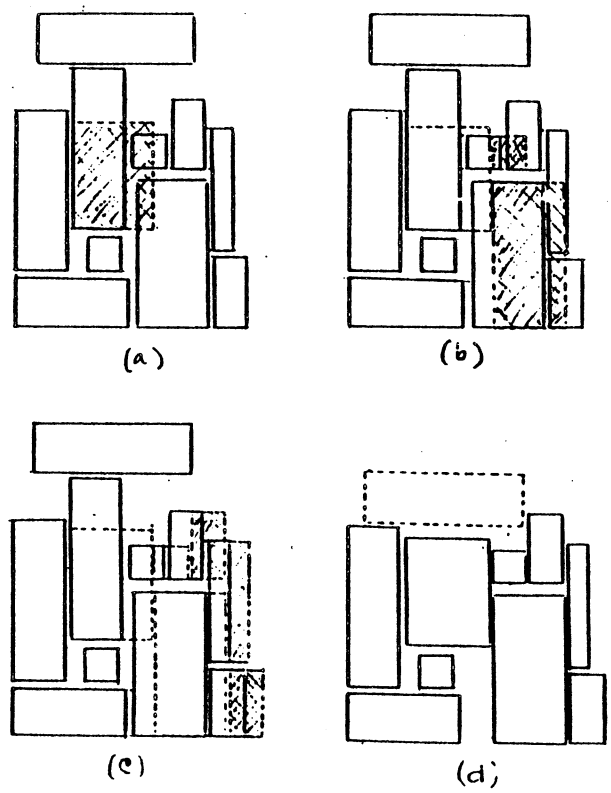


Fig IV-36. Exemple de propagation.

L'algorithme de propagation positive peut être comme le suivant :

1. Enumérer l'espace couvert par le bloc modifié (déplacé).
2. Déterminer les voisins dans la liste d'énumération.
3. Pour chaque bloc x voisins
   - marquer le déplacement du bloc x pour qu'il soit à l'extérieur de l'espace énuméré.
   - si la nouvelle position du bloc x dépasse sa zone libre, alors appliquer recursivement 1.
4. Mettre à jour la position des blocs marqués.

La complexité de la propagation positive est proportionnelle à D(N+p*p). Où D est le nombre des blocs déplacés, N est le nombre total des blocs et p est le nombre moyen des blocs dans l'espace énuméré.

Propagation négative.

Dans la propagation négative, on détermine d'abord les voisins d'un bloc, ensuite pour chaque bloc voisin, on détermine leurs voisins dans la direction opposée. Pour cette propagation, l'énumération

de l'espace ne peut pas être appliquée, il faut utiliser une projection.

L'algorithme de propagation négative est le suivant :

1. Faire la projection du bloc modifié (déplacé) dans la direction approriée.
2. Déterminer les blocs voisins.
3. Pour chaque bloc x voisins :
   - Déterminer les voisins de x dans la direction opposée.
   - Déterminer la limite de la zone libre dans cette direction
   - Déplacer x jusqu'à cette limite, si c'est possible. Appliquer 1) recursivement pour x.

La propagation négative se termine, lorsque tous les blocs voisins ne peuvent plus être déplacés.
Cette propagation est deux fois plus complexe que la propagation positive.

4.2.4.5. Mécanismes de base.

4.2.4.5.1. Création d'un bloc.

La création d'un bloc nécessite seulement une vérification sur la place désignée. Lorsque une telle place est libre, l'étape suivante ne concerne que l'insertion du bloc créé, dans la liste des blocs. Elle est très simple, mais elle nécessite un balayage total.

4.2.4.5.2. Suppression d'un bloc.

La suppression est l'opération la plus simple. Il suffit d'enlever le bloc supprimée, de la liste total. Mais lorsqu'on veut remplir l'espace libéré, la complexité de cette opération est celle de la propagation négative.

4.2.4.6. Mécanisme d'estimation des canaux de routage.

Comme il n'y a aucun objet invisible utilisé, il n'est pas possible d'enchaîner l'espace libre autour de chaque bloc.

4.3. DISCUSSION.

Nous espérons que l'exposé des mécanismes de base, liés aux quatre représentations étudiées, a permi au lecteur de se faire un idée sur l'édition d'un plan de masse et des problèmes qui en découlent.

Dans ce paragraphe nous essayons d'analyser ces représentations en vue d'obtenir la représentation la "plus adaptée" à notre besoin.

Considérons d'abord la complexité de propagation pour chaque représentation. Celle-ci est montrée dans le tableau ci-dessous.

	interface	séparateur	tuile	liste
complexité				
: moyenne	$\sqrt{I}$	d+s	d(c+e+i)	d(N+p*p)
: extrême	N	N+S/2	d(N+C)	d(N+p*p)
objets invisibles utilisés.	$I \leq N$	$S \leq 2N$	$C \leq 3N$	0

Tableau 4-1. Récapitulatif sur la complexité de propagation et les objets invisibles utilisés.

où :

N est le nombre total de blocs
I est le nombre total d' interfaces
S est le nombre total de séparateurs
C est le nombre total de tuiles
d est le nombre des blocs déplacés
s est le nombre des séparateurs déplacés
c est le nombre moyen des tuiles autour d'un bloc
p est le nombre moyen des blocs dans la projection
e est la complexité de création
i est la complexité de suppression

Sur le tableau 4-1, on voit que les complexités moyennes et extrêmes s'élèvent vers la droite.

Une autre considération qui peut être intéressante à comparer, est la qualité de propagation. Nous entendons par qualité de

propagation, la possibilité de déplacement des blocs se trouvant dans la projection d'un bloc modifié (déplacé). Les représentations par tuile et liste sont sans doute les plus adéquates. Elles permettent de déplacer les "vrais voisins". Tandis que dans les représentations par interface et séparateur les voisins déplacés sont ceux situés de l'autre côté de l'interface ou du séparateur. Lorsque la "relation sémantique", relation indiquant que deux blocs doivent être toujours face à face, (par exemple dans le cas d'un aboutement), doit être conservée, alors la représentation par interface est la plus intéressante.

Comparaisons sur l'estimation des canaux de routage.

Les trois premières représentations étudiées permettent de dégager des informations pour cette estimation. Nous pensons que le mécanisme d'estimation est très adapté à la représentation par séparateur, car il permet des affectations de trajet d'interconnexions sur des transparences de bloc de façon très simple. Malheureusement, la recherche d'un trajet n'est pas toujours convergente dans un seul parcour de graphe.

Dans la représentation par tuile, le mécanisme d'estimation des canaux permet d'obtenir une précision supérieure. Malheureusement elle provoque une limitation quand à la modification des formes. Il n'est plus possible de faire un ajustement, lorsque l'estimation des canaux se fait. Autrement dit, l'estimation des canaux et l'ajustement des blocs ne peuvent se succéder.

Dans la représentation par interface, l'estimation est moins adéquate que celles liées aux deux répresentations précédentes. Mais elle permet l'ajustement des forme des blocs et l'estimation des canaux en alternance.

Comparaisons sur la complexité des algorithmes de base.

Le tableau ci-dessous présente un récapitulatif des mécanismes de

base liées à toutes les représentations étudiées.

Mécanisme	interface	séparateur	tuile	liste
. recherche				
- d'un point	$\sqrt{I}$	$\sqrt{(N+S)}/2$	$(\sqrt{N+C})/2$	N
- d'un espace	$\sqrt{I}$	N	c	N
- des voisins	4	1	(c+p*p)	N*p*p
. création	$\sqrt{I}$	S	c	N
. suppression	2	5	c	1

Tableau 4-2. Récapitulatif de la complexité des mécanismes de base pour chaque représentation. (les significations sont les mêmes que pour le tableau 4-1)

Dans ce tableau on voit que pour un mécanisme donné sa complecxité dans une représentation est inférieure à celle dans des autres représentations.

Il est intéressant de noter que la complexité des mécanismes dans la représentation par interface est généralement inférieure à celle des autres représentations.

Il est également intéressant de noter que dans la représentation par tuile, la plupart des opérations ne dépendent pas du nombre total de blocs. Autrement dit, les opérations sont basées sur les informations locales. Ce qui permet de réduire les côuts de parcours d'une structure. Cette réduction se voit particulièrement pour des configurations dans lesquelles le nombre de rectangles est très grande, par exemple, dans le cas de descriptions au niveau transistor.
Cependant, pour le nombre des blocs comme dans le cas du plan de masse, le côut d'application des informations locales est presque

équivalent aux autres cas. Ceci est notamment lié au côut pour gérer les objets invisibles (tuiles) dont le nombre est trois fois que celui les blocs fonctionnel.

La représentation par liste, est intéressante à utiliser lorsque le nombre des bloc est très faible. Cependant, il manque l'information nécessaire à l'estimation des canaux. Une combinaison avec la représentation par séparateur peut être intéressante. Pendant la session d'ajustement on applique la représentation par liste et pour l'estimation des canaux on utilise la représentation par séparateur. Mais dans ce cas l'ajustement de forme et l'estimation des canaux ne peuvent alterner.

Conclusion.

Nous avons montré que chaque représentation est supérieure aux autres pour certaines opérations. Il nous semble que la représentation par interface est suffisamment adaptée pour la conception de plan de masse basée sur des évaluations prédictives.

4.4. REPRESENTATION IMPLANTEE.

L'analyse faite au paragraphe précédant, nous a conduit à implanter un prototype de mécanisme d'édition de plan de masse basé sur la représentation par interface.

Ce choix est justifié, non seulement par les raisons présentées précédemment, mais aussi par les raisons suivantes que nous pensons propres à la représentation par interface.

Description des blocs de façon relative.

En général, la forme des blocs à placer est connue d'avance. Donc le placement est relativement facile. Dans notre cas, la forme initiale des blocs est à priori inconnue. L'ensemble des blocs à évaluer et placer peut être décrit sous une forme textuelle. Mais

il n'est pas possible dans la description textuelle, d'indiquer le positionnement de chaque bloc, autrement que de façon relative. C'est par exemple, dire que tel bloc doit être à droite de tel autre bloc. Tels blocs doivent être au nord de tels blocs. Ces positionnements relatifs ne sont possible que pour une représentation par interface.

La génération d'une description pour des outils d'assemblage de type LUBRICK [SCH-83].

L'outil d'assemblage des blocs du système CAPRI, LUBRICK, génère le trajet exact des interconnexions entres des blocs, à l'aide des opérateurs UP, CUP, CRUP, RIGHT, CRIGHT et CRRIGHT. Les blocs à interconnecter doivent être mis horizontalement ou verticalement. Les opérateurs UP, CUP, CRUP effectuent l'assemblage dans le sens vertical. Les opérateurs RIGHT, CRIGHT et CRRIGHT effectuent l'assemblage dans le sens horizontal.

La représentation interne par interface permet de générer directement la description d'entrée pour assembleur, car chaque interface correspond à un opérateur de l'assembleur.

L'homogènité de l'organisation topologique et fonctionnelle des blocs.

Elle permet une implantation des traitements de manière uniforme.

Chapitre 5

MECANISME DE COMMUNICATION

Dans le chapitre précédent, nous avons présenté la solution qui permet de surmontrer le problème d'incohérence topologique causé par des modifications sur la forme des blocs. Ces modifications sont faites afin d'obtenir la forme convenable d'un bloc dans son environnement. L'agent de cette modification est l'évaluateur associé au bloc.

L'évaluateur topologique doit être composé d'un module de contrôle et d'un ensemble de routines d'évaluation de surface, de forme et de connexions pour les principaux blocs qui constituent un circuit VLSI (figure V-1).

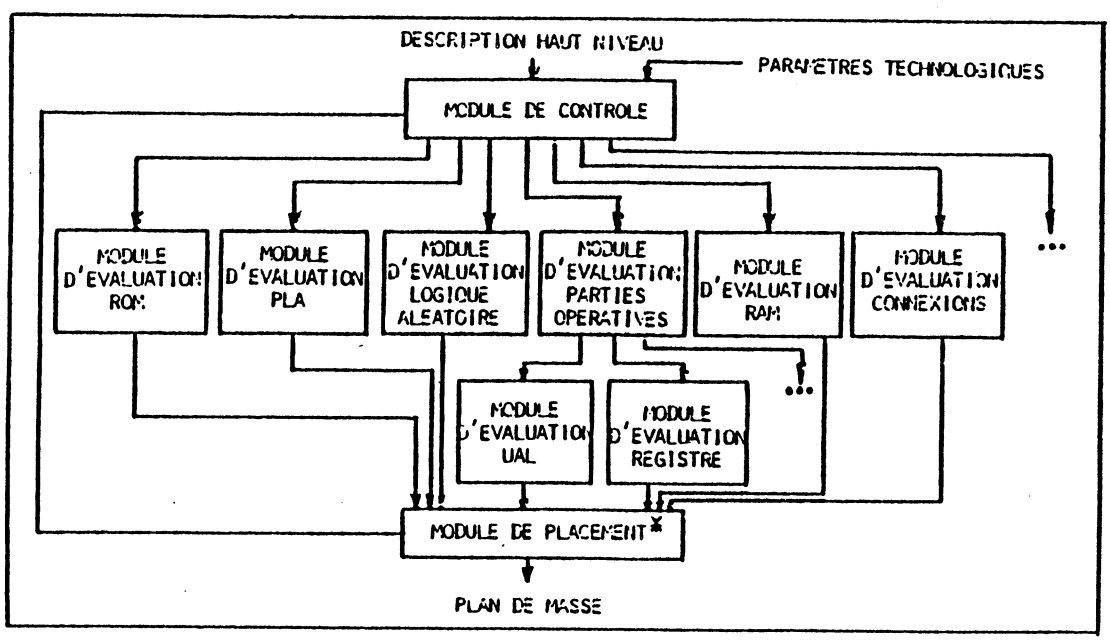


Fig V-1. Evaluateur topologique pour circuits VLSI [REI-83].

Le schéma ci-dessous illustre le principe d'un programme évaluateur.

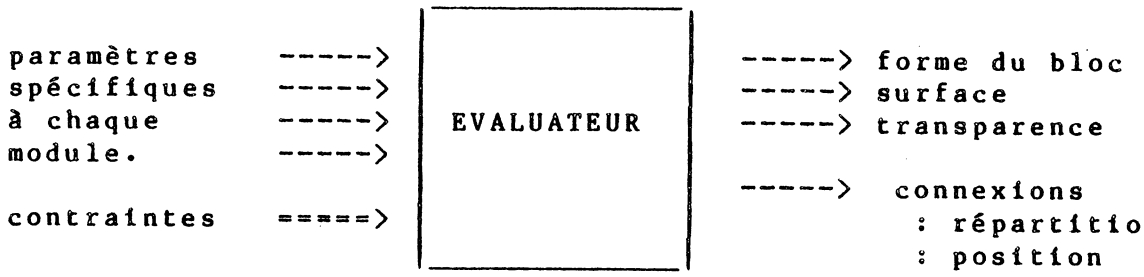


Fig V-2. Schéma de l'évaluateur.

Dans le système CAPRI, il existe actuellement plusieurs prototypes d'évaluateurs :

- l'évaluateur ROM [REI-83], [MAI-83].
- l'évaluateur PLA [REI-83].
- l'évaluateur LOGIQUE-ALEATOIRE [REI-83].

Ce chapitre est consacré au problème de la communication entre l'éditeur FLOPE et les évaluateurs. La communication est envisagée non seulement pour des évaluateurs existants, mais aussi pour de futurs évaluateurs.

Le premier paragraphe du chapitre introduit le problème. Un modèle de communication entre l'utilisateur, l'éditeur et les évaluateurs est présenté dans le deuxieme paragraphe. Enfin le troisième paragraphe contient des propositions afin de résoudre le problème évoqué.

5.1. PROBLEME.

Le problème posé dans la communication entre l'éditeur et les évaluateurs trouve son origine dans la diversité des paramètres spécifiques à chaque évaluateur. Ce problème concerne **le paramétrage des procédures et le nombre des paramètres liés à chaque procédure.**

Si l'on considère les évaluateurs existant comme une partie de l'éditeur, le problème n'apparait pas. Car toutes les particularisations peuvent être prises en compte dans la réalisation de

l'éditeur. Mais cela implique une restriction dans l'implantation de futurs évaluateurs. Il faut connaître les parties concernées de la structure de l'éditeur.

Notre but est d'éviter cela et de ne considérer que des évaluateurs indépendants de l'éditeur.
Les principes retenus sont les suivants :

- L'implantation d'un évaluateur dans l'environnement de l'éditeur est effectuée par une commande. Il n'y a pas de liaison préalable entre eux.

- L'implanteur ne déclare à l'éditeur que les paramètres spécifiques des évaluateurs.

5.2. MODELE DE COMMUNICATION.

Lors de l'évaluation d'un bloc, l'utilisateur fait appel aux paramètres permettant cette évaluation. L'éditeur fournit ces paramètres à partir des informations déclarées lors de la définition de l'évaluateur.

En se basant sur le schéma V-2, les paramètres spécifiques attachés à la déclaration sont groupés de la manière suivante :

- type de l'évaluateur
- nom du programme d'évaluation
- paramètres d'entrée
- paramètres de contrainte
- paramètres de sortie.

L'interaction entre l'utilisateur, l'éditeur et les évaluateurs peut être modélisée comme le montre la figure ci-dessous :

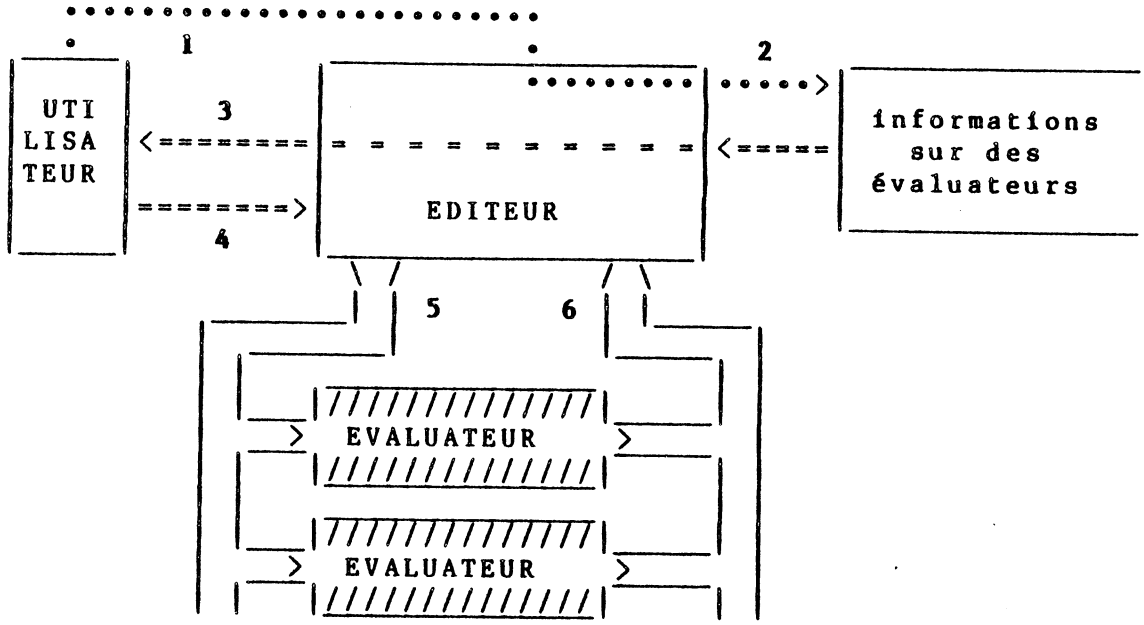


Fig V-3. Modèle d'interaction entre l'utilisateur, l'éditeur et les évaluateurs.

Les différentes communications possibles sont numérotées de 1 à 6 :

1. L'utilisateur demande une évaluation ou réévaluation du bloc X.
2. Recherche des paramètres d'évaluation (pour le bloc X), par l'éditeur.
3. Fourniture des paramètres nécessaires à l'évaluation du bloc X.
4. Valeur de paramètres, données par l'utilisateur à l'éditeur.
5. Envoi des données par l'éditeur à l'évaluateur concerné.
6. Récupération du resultats d'évaluation par l'éditeur.

Le problème du paramétrage évoqué au paragraphe précédent apparait aux étapes 5 et 6.

5.3. SOLUTION PROPOSEE.

Nous présentons trois possibilités de communication entre l'éditeur et les évaluateurs (étape 5 et 6). Chaque possibilité dépend en partie du langage de programmation utilisé, car la nature du problème se situe plutôt au niveau de la réalisation de l'éditeur. Ces possibilités sont :

- la communication via l'environnement du système hôte.
- la communication par fichier.
- la communication via l'environnement du langage.

5.3.1. Communication via l'environnement du système hôte.

Ici on utilise un programme d'interface permettant d'aiguiller l'appel sur l'évaluateur en fonction de la nature du bloc concerné. La figure ci-dessous schématise cette communication.

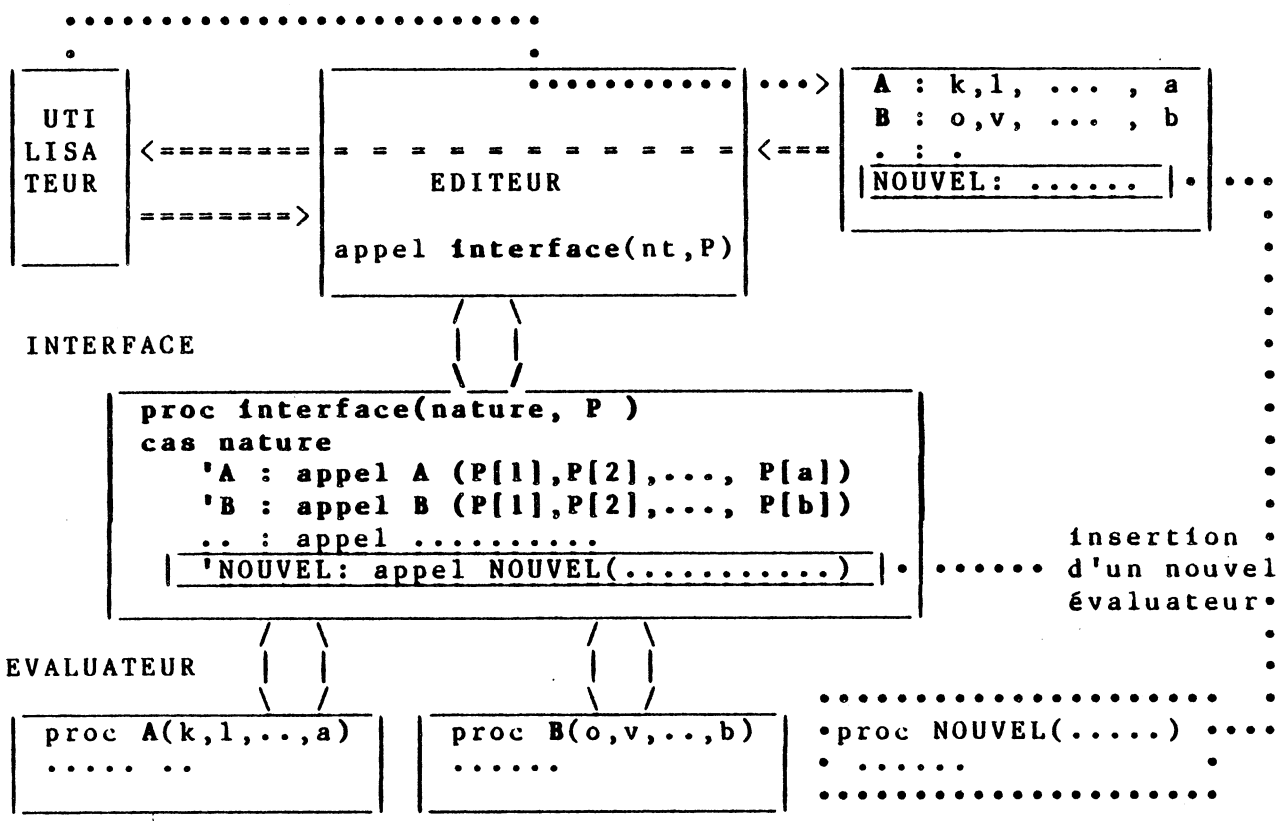


Fig V-4. Communication via l'environnement du système hôte.

Dans le schéma ci-dessus, l'éditeur envoie les données d'un bloc au programme d'interface qui sélectionne l'évaluateur approprié. Le programme d'interface contient toutes les données nécessaires à la sélection et l'activation d'un évaluateur. Il rend à l'éditeur le résultat de l'évaluation.

Dans ce type de communication, lors de l'insertion d'un nouvel évaluateur, il faut déclarer les paramètres spécifiques et mettre à jour le programme d'interface.

L'inconvénient d'une telle communication est qu'il n'est pas possible d'insérer automatiquement un nouvel évaluateur dans le programme d'interface à partir des informations contenues dans sa déclaration.

D'autre part, la communication ne peut être réalisée que dans l'environnement d'un système hôte qui permet une **édition de liens dynamique**.

5.3.2. Communication par fichier.

Une autre possibilité de communication, est l'utilisation des fichiers d'entrée-sortie. Elle est schématisée comme le montre la figure ci-dessous.

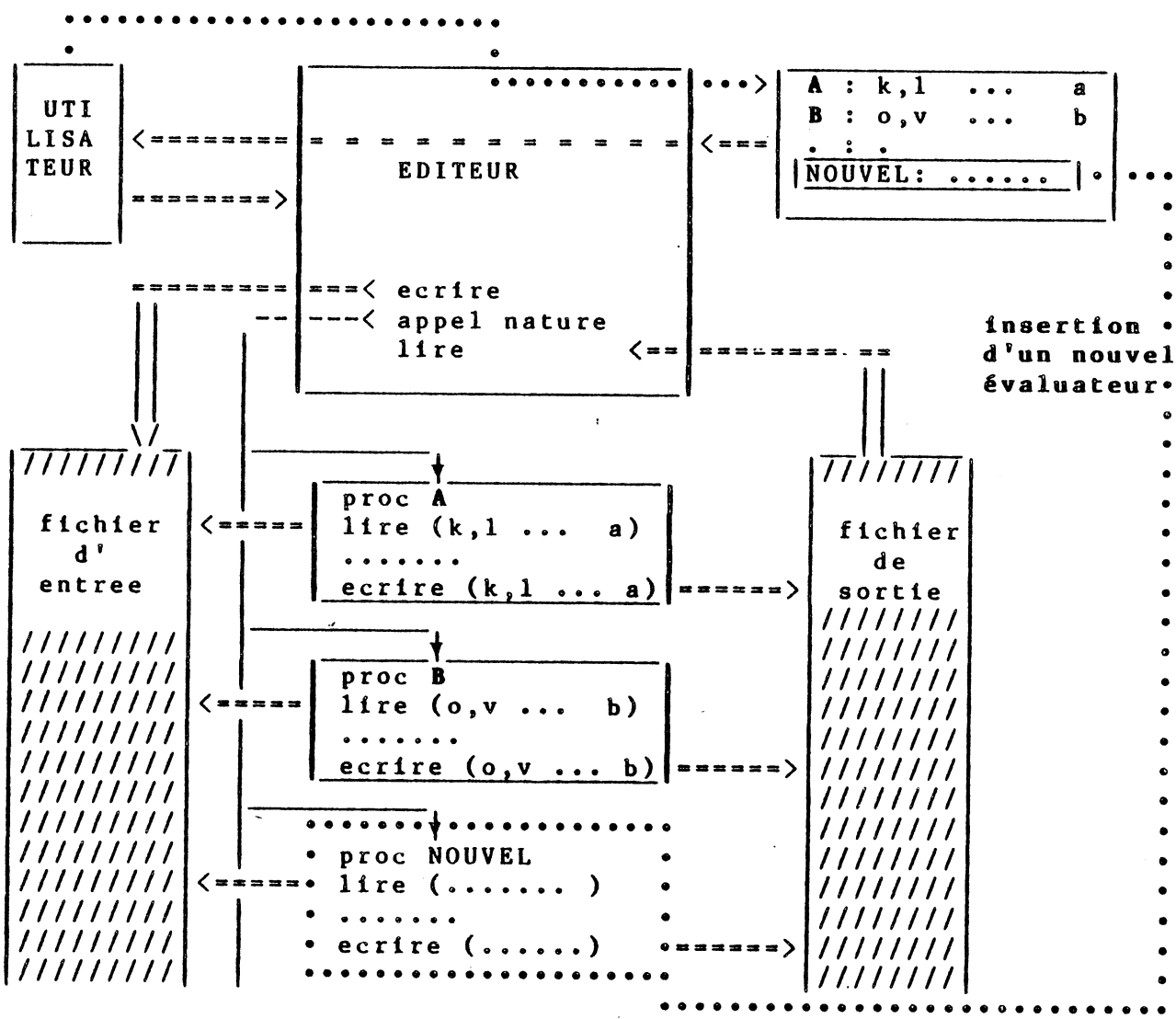


Fig V-5. Communication par fichier.

Sur le schéma ci-dessus, l'éditeur écrit les données d'évaluation d'un bloc sur le fichier d'entrée et ensuite fait appel à l'évaluateur approprié, sans paramètres. Cet évaluateur lit les données, les interprète et écrit le résultat dans le fichier de

sortie. L'éditeur lit ce résultat.

Dans ce type de communication, l'insertion d'un nouvel évaluateur peut être faite de façon automatique à partir des informations contenues dans sa déclaration. Le problème des paramètres d'appel n'apparait pas.

Cette communication peut être implantée dans la plupart des systèmes hôte.

5.3.3. Communication via l'environnement fournis par le langage utilisé.

Dans les deux solutions précédentes, il est toujours nécessaire de remettre à jour le programme d'interface de communication quand on insère un nouvel évaluateur. Pour la deuxième solution, cette opération est prise en charge par l'éditeur.

La troisième solution que nous présentons ici, ne nécessite pas de mise à jour du programme d'interface. Un nouvel évaluateur à insérer ne nécessite que sa déclaration. Malheureusement, cette solution n'est possible que dans un environnement de programmation où un programme peut être traité comme des données. C'est par exemple le cas d'un environnement de programmation basé sur le langage LISP ou le langage PROLOG.
La figure ci-dessous montre un schéma de communication basé sur cette possibilité.

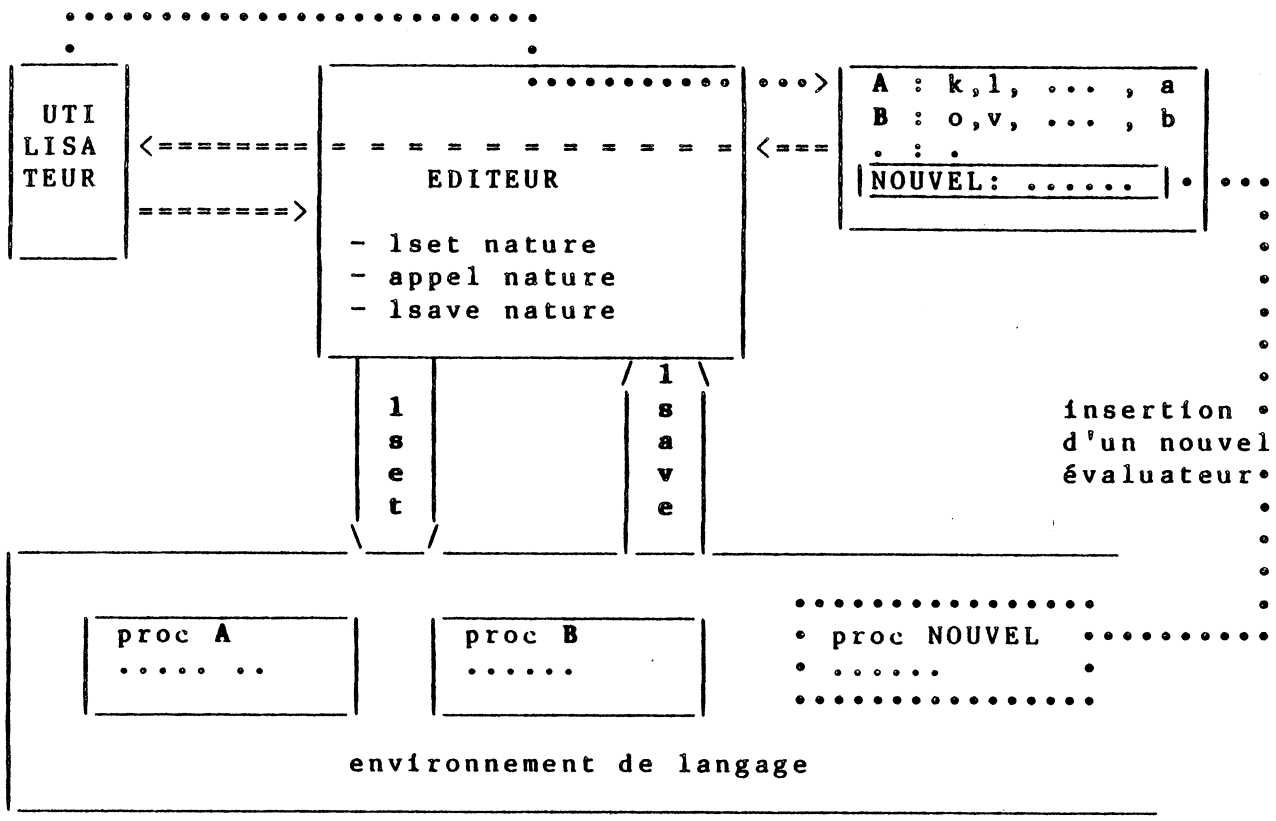


Fig V-6. Communication via l'environnement fourni par le langage.

Avant d'appeler l'évaluateur concerné, l'éditeur affecte les données dont l'évaluateur a besoin à l'environnement de programmation, à l'aide de l'instruction **lset**.

A la fin de l'évaluation, l'éditeur récupère les résultats, à l'aide de l'instruction **lsave**.

Chapitre 6

PRESENTATION EXTERNE DU SYSTEME FLOPE.

6.1. FONCTION DU SYSTEME.

Ce système aide à la construction, la modification et l'optimisation du plan de masse de circuits intégrés. Son action principale est d'évaluer la surface des blocs (en appelant des outils d'évaluation), de les placer et de les ajuster.

6.2. REPRESENTATION DES BLOCS.

Un bloc du plan de masse est représenté par un polygone et, éventuellement, des connecteurs. Dans la version actuelle les formes sont limitées aux rectangles.

Un bloc est défini par les élements suivants :

NOM : sert à identifier un bloc.

NATURE : désigne le type d'évaluateur appliqué.

DEFORMABILITE : représente l'état actuel du bloc.

PARAMETRES D'EVALUATION : précise les paramètres d'évaluation d'un bloc en fonction de sa nature.

GEOMETRIE : indique la forme du bloc, soit un rectangle soit un polygone.

CONNECTEUR : contient les informations sur les connecteurs.

TRANSPARENCE : indique le nombre de connexions pouvant traverse un bloc.

POSITION : indique les coordonnées de l'origine du bloc (en bas à gauche) sur le plan de masse.

CONTENU : indique la liste des blocs contenus dans un bloc.

ATTITUDE : indique la transformation géométrique subie par un bloc. Elle peut être une opération de rotation ou de symétrie.

6.3. VUE GENERALE.

Une vue générale du système, contient principalement 5 parties, comme le montre la figure VI-1. Ces parties sont :

- la structuration
- la description externe
- l'interface avec les modules d'évaluation
- la représentation graphique
- les commandes utilisateur.

Les descriptions externes données par le concepteur, vont être évaluées par les évaluateurs et le transit entre FLOPE et ces évaluateurs se fera au moyen d'une interface. En utilisant les commandes d'édition, le concepteur peut de manière interactive modifier, ajouter ou supprimer un bloc. Il est également possible de réorganiser le plan de masse, de sauvegarder plusieurs version , etc ...

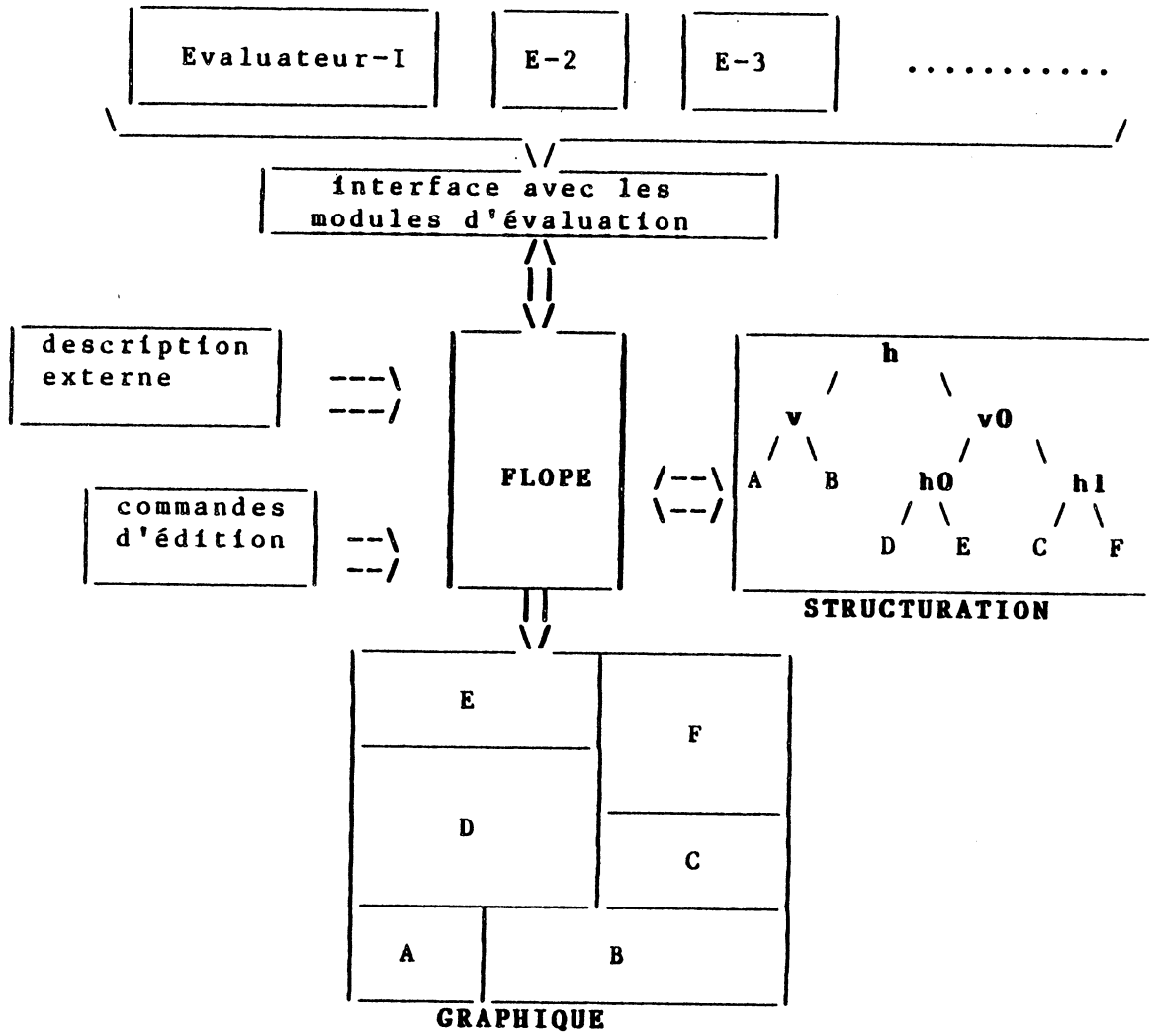


Fig VI-1. Vue générale du système FLOPE.

6.3.1. Structuration.

La structuration en figure VI-1 représente une décomposition hiérarchique correspondant à la description externe d'un circuit.

Deux types de blocs sont distingués : le bloc feuille (terminal) de l'arbre hiérachique qui contient un bloc fonctionnel, et le bloc composé (non terminal) qui contient d'autres blocs.

Bloc feuille. La forme d'un bloc feuille est controlée soit par la fonction standard de déformabilité, soit par un programme d'évaluation.

Bloc composé. La forme d'un bloc composé est un rectangle d'encombrement lié aux blocs le constituant. On définit deux types de blocs composés : les composés horizontaux et les composés verticaux.

Composés horizontaux.
Les constituants d'un bloc composé horizontal sont positionnés sur le plan de masse, de gauche à droite avec éventuellement un offset associé à chaque bloc.

Composés verticaux.
Les constituants d'un bloc composé vertical sont positionnés sur le plan de masse, de bas en haut avec éventuellement l'offset associé à chaque bloc.

Exemple :

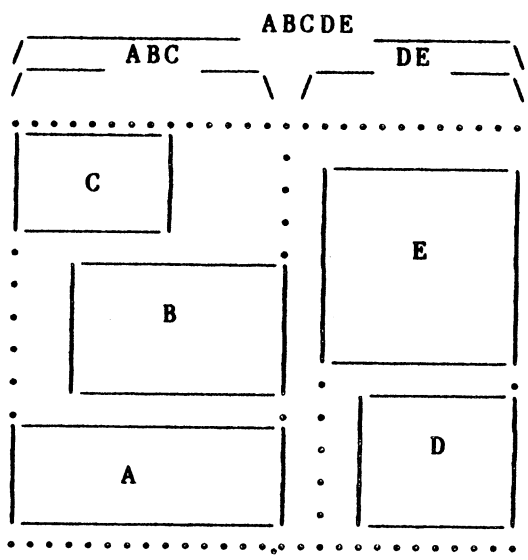


Fig VI-2. Exemple de blocs composés verticaux et horizontaux dans une structure.

Sur la figure VI-2 le bloc composé horizontal ABCDE est constitué par les blocs composés verticaux ABC et DE.

6.3.2. Description externe.

Au debut de la construction du plan de masse, le concepteur ne connait pas, à priori, la forme des blocs. C'est les évaluateurs qui les dimensionneront en fonction de leur nature.

La description externe représentant un circuit doit contenir, pour chaque bloc donné, trois informations :

- la nature du bloc
- le nom du bloc
- le type de composition.

La syntaxe complète d'une description externe est la suivante :

```
<BLOC> ::= ( <TYPE-DE-COMPOSITION>
             <NOM-DE-BLOC>
             <NATURE-ET-PARAMETRES>
             <LISTE-DE-BLOC-FILS>     )

<LISTE-DE-BLOC-FILS> ::= <BLOC>
                       | <LISTE-DE-BLOC-FILS>
                       | nil

<TYPE-DE-COMPOSITION> ::= h | v | *

<NOM-DE-BLOC> ::= chaine-de-caractéres

<NATURE-ET-PARAMETRES> ::= PLA ( <LIST-VAR-PLA> )
                         | ROM ( <LIST-VAR-ROM> )
                         | LOGAL ( <LIST-VAR-LOGAL> )
                         | LIBRE ( <LIST-VAR-LIBRE> )
                         | <DEFINI> ( <LIST-VAR-DEFINI> )

<LIST-VAR-PLA>    ::= (ne chiffre ) (nm chiffre )
                      (nn chiffre )

<LIST-VAR-ROM>    ::= (nb chiffre ) (nm chiffre )

<LIST-VAR-LOGAL> ::= (ntran chiffre )

<LIST-VAR-LIBRE> ::= (dx chiffre )   (dy chiffre )

<DEFINI> ::= nature du module externe.

<LIST-VAR-DEFINI> ::= liste de couples contenant les
                      variables liées au module externe
                      et leur valeur.
```

La figure VI-3 donne une description contenant trois blocs fonctionnels.

```
(v bloc2 EN
   (h bloc1 EN
      (* bloc-rom ROM  (nm 20) (nb 40) )
      (* bloc-pla PLA (ne 30) (nm 24) (nn 16) )
   )
   (* bloc-log LOGAL (ntram 50) )
)
```

Fig VI-3. Exemple de description externe.

ROM, PLA, LOGAL (LOGique ALeatoire) et EN (encombrement) sont des natures de blocs. bloc2, bloc1, bloc-rom, bloc-pla et bloc-log sont les noms des blocs. h et v sont les types des blocs composés. La marque * à la place du type d'un bloc composé indique la présence d'un bloc feuille (fonctionnel). La signification de la description de la figure VI-3 est la suivante : le bloc2 est constitué, de bas en haut, du bloc1 et du bloc-log. Le bloc1 est, lui-même, constitué, de gauche à droite, du bloc-rom et du bloc-pla et du bloc-rom. La figure VI-4 clarifie ce placement relatif.

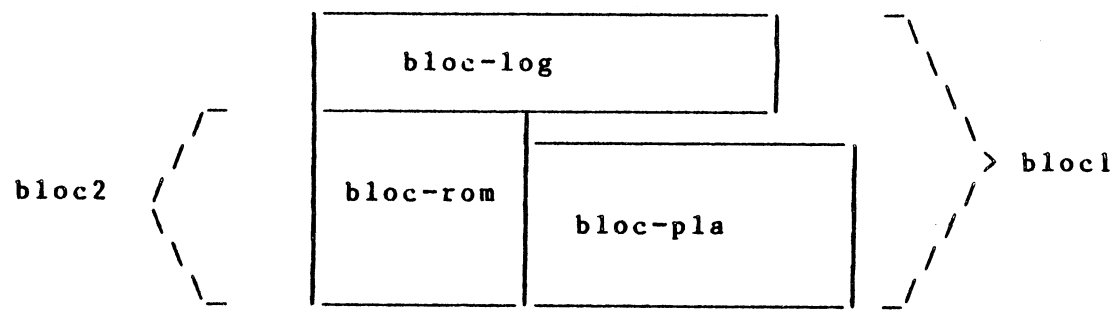


Fig VI-4. Placement relatif correspondant à la figure VI-3.

6.3.3. Interface entre l'éditeur et les modules d'évaluation.

Pour évaluer un bloc, FLOPE fait appel au module d'évaluation caractérisé par la nature du bloc.

Chaque module d'évaluation est spécialisé dans l'estimation des caractéristiques topologiques :

- la forme et la surface d'un bloc
- la disposition des connecteurs

Un module d'évaluation ne peut être appelé qu' après avoir été déclaré dans l'éditeur FLOPE. Donc les informations qui transitent entre FLOPE et les évaluateur sont :

- le type de l'évaluateur
- les paramètres spécifiques à chaque évaluateur
- le niveau d'évaluation choisi

Les modules d'évaluation sont indépendants de l'éditeur FLOPE. Le concepteur peut définir ses propres modules suivant les besoins de son travail et l'évaluation topologique qu'il souhaite réaliser. Il lui suffit de déclarer à l'éditeur FLOPE les informations d'interface. La figure VI-5 illustre la syntaxe d'une déclaration.

```
(defmodule
   (programme (...))
   (nature  (...))
   (niveau  (...))
   (entrées (...))
   (sorties (...))
   (contraintes   (...))
)
```

Fig VI-5. Syntaxe de la déclaration d'un évaluateur.

programme :
indique le nom du programme d'évaluation. Ce programme doit

être écrit sous le langage LE_LISP.

nature :
indique la nature du bloc à évaluer par le programme lié à l'évaluateur.

niveau :
indique le niveau d'évalution du programme ci-dessus. Les niveaux sont : flou, mou et dur.

entrées :
contient la liste des variables, entrées pour le programme d'évaluation.

sorties:
contient la liste des variables retournées par le programme. Pour la dimensionnement du bloc, le programme doit utiliser les variables **dx** pour la largeur et **dy** pour la hauteur du bloc.

contraintes :
contient la liste des variables sur les contraintes. Deux variables ont été définies par l'éditeur FLOPE : **cdx** et **cdy** qui sont utilisées pour les contraintes sur les dimensions d'un bloc. cdx est un contrainte sur la largeur (direction des X) et cdy est une contrainte sur la hauteur (direction des Y).

Des exemples de déclaration d'un module sont donnés en annexe 2.

6.3.4. Représentation graphique.

Le résultat d'une évaluation ou d'une modification est visualisé sur l'écran qui est organisé comme le montre la figure VI-6.

La visualisation est incrémentale. Dès qu'une opération modifie une partie de la figure, le système calcule la nouvelle figure à visualiser, en ne modifiant sur l'écran que les parties du plan de masse ayant évolué.

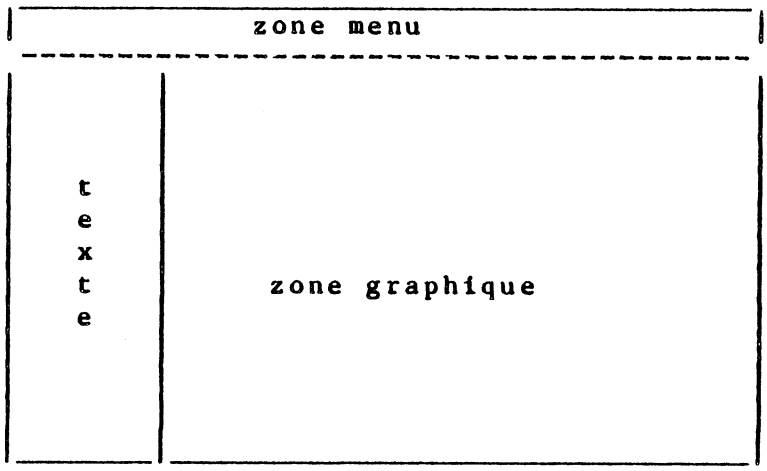


Fig VI-6. Division de l'écran.

La figure présentée à l'écran montre toujours les blocs feuilles d'un plan de masse.

La modification d'un bloc peut concerner sa forme ou son attitude. Cette modification a automatiquement des effets sur les blocs voisins. La figure VI-7 montre une exemple de modification topologique.

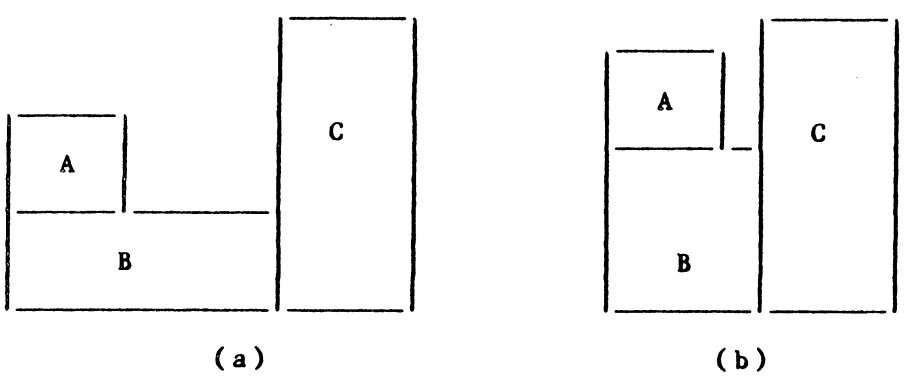


Fig VI-7. Effets d'une modification topologique.

La figure VI-7a illustre les positions des blocs A, B et C avant modification du bloc B, et la figure VI-7b après modification.

6.3.5. Commandes "utilisateur".

Les commandes "utilisateur" s'appliquent généralement au bloc courant : un bloc qui vient d'e être crée où désigné. Il est possible de désigner les blocs de plusieur manières :

• désignation directe - par le curseur - : l'utilisateur valide la position du curseur sur l'écran, et l'éditeur détermine le bloc feuille couvrant cette position. Il devient alors le bloc courant.

• désignation relative : A partir du bloc courant on cherche le bloc

- précédent ou suivant à l'intérieur du bloc composé père
- englobant (bloc père).
- racine.
- courant précédent (retour en arrière)

• désignation par nom d'un bloc. Il est également possible de marquer un bloc (i.e. lui donner un nom), afin de le désigner ultérieurement par son nom de marquage.

Les commandes utilisateur sont groupées en :

- commandes d'évaluation : pour évaluer ou ré-évaluer la forme d'un bloc.

- commandes de manipulation : pour créer, supprimer, substituer, déplacer, tourner un bloc feuille ou composé.

- commandes diverses : pour sauvegarder, restaurer, lister, visualiser, etc..., une configuration.

La description détaillée de chaque commande est donnée en annexe 1.

Chapitre 7

REALISATION

Ce chapitre présente les deux points suivants :

- la stratégie de réalisation de l'éditeur.
- l'implantation de l'éditeur en langage CEYX.

7.1. STRATEGIE DE REALISATION DE L'EDITEUR.

Les critères qui ont guidé la conception et le développement de ce système sont :

- **La portabilité :**
  Elle permet la diffusion du système et repose sur le choix d'un langage de programmation portable, qui est dans notre cas, le langage CEYX.

- **L'extensibilité :**
  Elle indique les possibilités d'ajout d'autres systèmes et d'extension du logiciel.

L'architecture interne du système FLOPE est schématisée dans la figure VII-1.
Le système est construit selon une architecture modulaire : chaque module du système a une fonction spécialisée, et peut être aisément remplacé ou modifié sans que cela nécessite la remise en cause des autres modules.

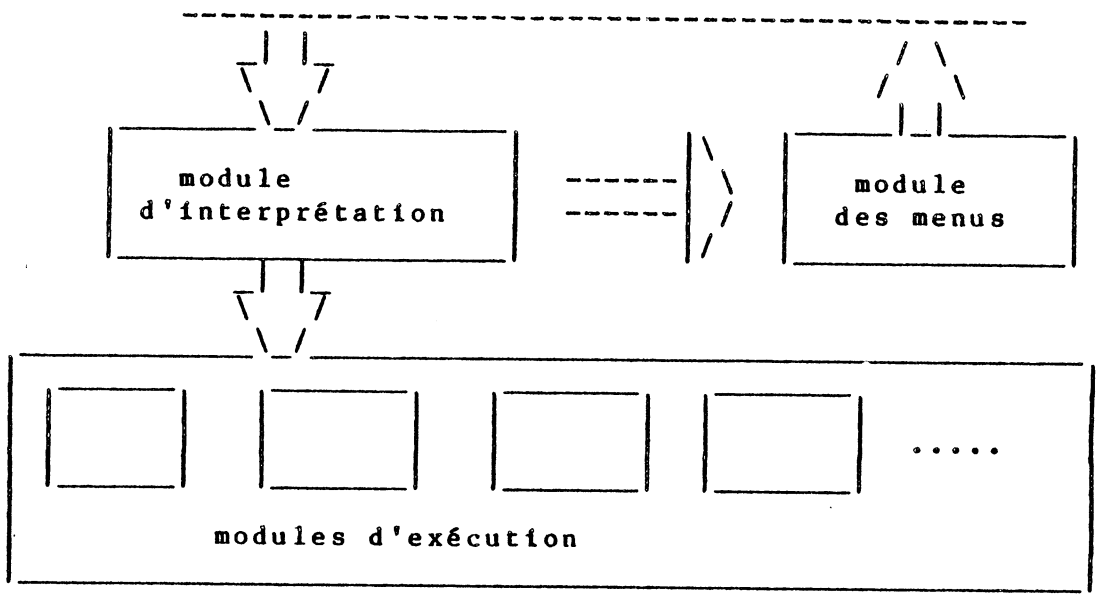


Fig VII-1. Architecture interne du FLOPE.

Il y a trois modules :

- le module d'interprétation
- le module des menus
- le modules d'exécution.

Le système dialogue avec l'utilisateur par le bias des modules d'interprétation et le module menu. Le module menu gère une structure de menu, tandis que le module d'interprétation prend en charge les commandes de l'utilisateur et détermine le module d'exécution concerné. Chaque module d'exécution est associé à une opération ou une commande utilisateur.

La structure de menu est décrite dans un arbre de profondeur 2. Chaque commande est constituée par deux caractères. Le premier caractère indique une commande sur le deuxième niveau dans l'arbre des menus. Alors que le deuxième caractère corespond à une opération d'édition. Après l'exécution d'une commande, le module d'interprétation revient au premier niveau du menu.

Dans l'utilisation d'un éditeur, l'utilisateur n'utilise fréquement que certaines opérations. Il est donc souhaitable de prévoir un

mécanisme de chargement dynamique des modules exécutables, car en court d'édition les modules non utilisés n'ont pas à résider en mémoire principale.

7.2. IMPLANTATION EN LANGAGE CEYX.

Le langage CEYX [HUL-83] est une extension du langage LE_LISP [CHA-83] destiné à apporter à ce langage toutes les facilités des définitions et traitements d'objets, enregistrements de type record en PASCAL. L'objet défini dans le langage CEYX est appelé **Classe.** Les opérations qui manipulent l'instance d'une classe sont appelées **fonctions sémantiques** d'une telle classe.

Les caractéristiques intéressantes de ce langage sont :

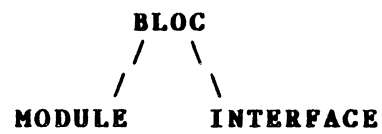
- la facilité d'extension d'une classe. Une nouvelle classe peut être construite à partir d'une autre classe déjà définie. Une classe ainsi construite est appelée **sous-classe.**

- L'héritage des opérations. Les fonctions sémantiques d'une classe sont valables pour toutes sous-classes qui sont construites par extension de cette classe.

Ces caractéristiques coincident aux besoins de nos mécanismes où les différents objets nécessitent une extension dynamique, notamment dans le cadre des communications avec les futurs évaluateurs. (cf. chapitre 5)

7.2.1. Structures de données.

L'objet de base qui est décrit au chapitre précédent, est implanté en CEYX dans une classe BLOC. Cette classe possède deux sous-classes : MODULE et INTERFACE.



La classe BLOC est décrite comme suit :

```
(deftclass  BLOC
        xor         entier
        yor         entier
        dix         entier
        diy         entier
        attitude    entier )
```

classe définissant les coordonnées, la dimension et la transformation (rotation et/ou symétrie) d'un bloc.

La sous-classe MODULE est définie comme suit :

```
(deftclass (BLOC):MODULE
        nom                 symbol
        nature              symbol
        paramétres          (List entier)
        déformabilité       entier
        transparence        entier
        connecteur          CONNEX
        père                INTERFACE  )
```

Ces champs ont la signification suivante :

- nom est l'identificateur de la figure qui est donné par l'utilisateur au moment de la création d'un bloc, afin de reconnaître les différentes figures.

- nature est le type de bloc associé à une évaluateur.

- paramètres est une liste des paramètres permettant l'évaluation et réévaluation d'un bloc

- déformabilité est un indicateur donnant le niveau d'évaluation.

- transparence est un entier représentant le nombre des connexions qui peut lui traverser.

- connecteur est un champs contenant les informations de connexion. Ces informations sont contenues dans un enregistrement de type CONNEX, dont la structure est la suivante :

```
(deflrecord CONNEX
        nom     symbol
        xcon    entier
        ycon    entier  )
```

- père est le pointeur sur l'interface.

La sous-classe INTERFACE est codée de la manière suivante :

```
(deftclass {BLOC}:INTERFACE
        type        symbol
        contenu     (list MODULE)
        trajet      (list symbol)
        père        INTERFACE        )
```

Les champs ont pour signification :

- type est un symbole indiquant l'interface verticale ou horizontale.

- contenu est la liste des blocs descendance.

- trajet est la liste des blocs qui utilisent cette interface pour leurs interconnexions.

7.2.2. Fonctions sémantiques.

Nous ne décrivons que les fontions sémantiques principales associées aux blocs. Ces sont des fonctions :

- Créer
- Détacher
- Attacher
- Modifier

Fonction Créer.

Cette fonction effectue les actions suivantes :

- faire appel à l'utilisateur afin qu'il donne les valeurs des paramètres pour le bloc à créer.
- invoquer l'évaluateur associé à un tel bloc.
- générer un instance contenant les résultats d'évaluation.
- passer à la fonction Attacher afin de placer l'instance dans la structure de données.

Fonction Détacher.

Cette fonction enleve de la structure un bloc indiqué. Une propagation négative est éventuellement appliquée.
Le bloc enlevé ne disparaît pas de l'environnement. Il est gardé dans une structure provisoire, appelée PILE.

Fonction Attacher.

Cette fonction évalue la position du bloc inséré provenant de la création ou de la structure provisoire. Elle crée une interface entre le bloc repéré et le bloc placé. Une propagation positive est éventuellement appliquée.

Fonction Modifier.

Cette fonction fait appel à l'utilisateur afin qu'il donne les valeurs des contraintes associées au bloc modifié. Elle fait ensuite :

- la vérification des contraintes données.
- la réévaluation, en faisant appel à l'évaluateur associé.
- la mise à jour de l'instance du bloc, à l'aide des résultats de réévaluation.
- la propagation si cela est nécessaire.

CONCLUSION

La conception et la réalisation d'un outil d'aide à la construction du plan de masse, ont été menées à bien dans le cadre de la méthodologie de conception de circuits VLSI **CAPRI**, dont l'une des stratégies principales est d'anticiper la conception détaillée en vue d'obtenir une topologie globale beaucoup plus régulière et optimisée ceci en un temps raisonnable.

La mise en ouvre de la stratégie d'optimisation basée sur la déformabilité des blocs a posé un problème de cohérence topologique. La plus grande partie de ce travail y est consacrée. Pour surmonter ce problème nous avons développé un mécanisme de propagation. Dans le but d'obtenir une implémentation efficace et acceptable de ce mécanisme, nous avons étudié plusieurs représentations internes du plan de masse.

La représentation interne choisie permet notamment de représenter à la fois le découpage fonctionnel et le découpage topologique en une seule hiérarchie.

L'autre problème à résoudre est celui de la communication avec de futurs évaluateurs. Les caractéristiques de ces derniers sont, à priori, très difficiles à prédire. Pour cela nous avons étudié plusieurs solutions en se basant sur un modèle de communication.

Le type de communication utilisé permet à l'utilisateur de définir son propre évaluateur.

BIBLIOGRAPHIE

[ANC-82a] F. ANCEAU, R. REIS :
"Complex Integrated Circuit Design Stragegy"
IEEE Journal of Solid State Circuits, Vol 17, No. 3,
pp. 459-464, Juin 1982.

[ANC-82b] F. ANCEAU :
"CAPRI : A Silicon Compiler for VLSI circuits Specified by Algorithms"
Proc. of the Advance Course on VLSI Architecture,
Université de Bristol, Royaume Uni,
19-30 Juillet 1982.

[ANC-83a] F. ANCEAU :
"Layout Strategy for NMOS-CMOS VLSI"
Proc. of 20th Design Automation Conference, 1983
pp. 705-708.

[ANC-83b] F.ANCEAU :
"CAPRI: A Design Methodology and a Silicon Compiler for VLSI"
Third Caltech Conference on VLSI, 1983
pp.15-31.

[AYR-79] R. AYRES :
"Silicon Compilation - A Hierarchical Use of PLAs"
Proc. of the First CALTECH Conference on VLSI, 1979
pp. 311-326.

[BOZ-84] J.J. BOZEK :
"**MUSIC: Management and Editing of Functional Design**"
University of Illinois at Urbana-Champaign,
Departement of Computer Science Report
UIUCDCS-R-84-1160, 1984.

[CHA-83] J. CHAILLOUX :
"**LE_LISP 80 version 12, le manuel de référence**"
rapport technique INRIA no 27, Juillet 1983.

[CHI-83] T. CHIBA, N. KUBO, T. KAMBE, T. INUFUSHI, I. NISHIOKA and S. KIMURA :
"**SHARP: A CAD System for VLSI**"
Proc. of 20th Design Automation Conference, 1983
pp. 532-353.

[CHU-84] S. CHUQUILLANQUI :
"**Une Nouvelle Approche pour l'Optimisation Topologique et l'Automatisation du dessin des masques de PLA complexe**"
Thèse Docyeur-Ingénieur, INPG,
15 Octobre 1984.

[DON-80] W. E. DONATH :
"**Complexity Theory and Design Automation**"
Proc. 17th Design Automation Conference, 1980
pp. 412-419.

[GUY-83] A. GUYOT, R. REIS, I. SUPRIANA :
"**FLOPE: Editeur du plan de masse des circuits intégrés**"
Rapport de Recherche no. 333,
IMAG, 1983.

[HAS-82] J.E. HASSET :
"**Automated Layout in ASHLAR: An Approach to the Problems of 'General Cell' Layout for VLSI**"
Proc. 19th Design Automation Conference, 1982
pp. 777-784.

[HUL-83] J.M. HULLOT :
"CEYX: A Multiformalism Programming Environnement"
IFIF83, R.E.A. Masson (ed), North Holland, Paris 1983.

[JER-83] A.A. JERRAYA :
"Une Nouvelle Approche pour la Vérification des masques des Circuits VLSI"
Thèse Docteur-Ingénieur, INPG,
24 Novembre 1983.

[LEB-83] André LEBLOND :
"CAF: A Computer-assisted floor-planning tool"
Proc. of 20th Design Automation Conference, 1983
pp. 747-753.

[MAI-83] F. MAILHOT :
"Evaluation d'une mémoire morte et de ses connexions"
Rapport de stage de DEA Microélectronique,
Juin 1983.

[MAL-82] K. MALING, S.H. MUELLER, W.R. HELLER :
"On Finding Most Optimal Rectangular Package Plans"
Proc. of 19th Design Automation Conference, 1982
pp.663-670.

[MAL-83] R. MALLADI, G. SERRERO, A. VERDILLON :
"Automatic Placement of Rectangular blocks with the Interconnexion Channel"
Proc. of 18th Design Automation Conference, 1981
pp. 419-425.

[MAR-83] MARINE et al :
"IRENE : Un Langange de Description des Circuits Intégrés logiques",
Rapport de Recherche no. 356,
IMAG, 1983.

[MAT-84] J.M. MATA :
"A Methodology for VLSI Design and A Constraint-based Layout Language"
Ph.D. Thesis, Princerton University,
October 1984.

[MEA-80] C. MEAD & L. CONWAY :
"Introduction to VLSI Systems"
Addison - Wesley Publishing Company, 1980

[MEY-82] D. MEYET :
"Evaluation Topologique pour VLSI"
Rapport d'ingenieur, ENSERG, Juin 1983 .

[OTT-82] R.H.J.M. OTTEN :
"Automatic Floorplan Design"
Proc. 19th Design Automation Conference, 1982
pp.261-267.

[PRE-79] B.T PREAS & W.M. van CLEEMPUT :
"Placement Algoritms for Arbitrarily Shaped Blocks"
Proc. 16th Design Automation Conference, 1979
pp. 474-480.

[REI-82] R. REIS:
"TESS: A Topological Evaluator Tool"
ICCC, 1982
pp. 539-542.

[REI-83] R. REIS :
"TESS: Evaluateur Topologique Prédictif pour la Génération Automatique des Plans de Masse de Circuit VLSI",
These de Docteur-Ingenieur, INPG,
Janvier 1983.

[REV-83] M.C. REVETT and P.A. IVEY :
"ASTRA: A CAD System to Support a Structured Approach to IC Design"
VLSI '83
pp. 413-422.

[SAH-80] S. SAHNI et A. BHATT :
"The Complexity of Design Automation Problems"
Proc. 17th Design Automation Conference, 1980
pp. 402-411

[SAU-84] G. SAUCIER, A. BELLON, J.F. PAILLOTIN, J.TSITSIMIS, A. JANATH, P. CHAISEMARTIN :
"Classification and Comparaison of Different Placement Strategies"
Proc. 21th Design Automation Conference, 1984
pp. 745-750.

[SCH-83] J.P. SCHOELLKOPF :
"LUBRICK : A Silicon Compiler and its Application to Data Path Design for FISC"
ESSIRC 83,
pp.435-446.

[SOU-80] J. SOUKUP, J. ROYLE :
"Cell Map Representation for Hierarchical Layout"
Proc. 17th Design Automation Conference, 1980
pp. 591-594.

[SUP-83] I. SUPRIANA :
"FLOPE : Un éditeur graphique de plan de masse de Circuits Intégrés"
Rapport de DEA Informatique, INPG,
Juin 1983.

[SUZ-81] A. SUZIM :
"Etude des Parties Opératives à Element Modulaire pour Processeurs Monolithiques"
Thèse Docteur-Ingénieur, INPG,
November 1981.

[SZE-80] A.A. SZEPIENIEC et al :
"The Geneological Approach to the Layout Problem"
Proc. 17th Design Automation Conference, 1980
pp. 535-542.

[TRI-82] S. TRIMBERGER and J. ROWSON :
"RIOT : A Simple Graphical Chip Assembly Tool"
Microelectronics 1982.
pp. 32-36.

[OUS-84] J.K. OUSTERHOUT, G.T. HAMACHI, R.N. MAYO, W.S. SCOTT and G.S. TAYLOR :
"MAGIC: A VLSI Layout System"
Proc. 21th Design Automation Conference, 1984
pp. 152-159.

ANNEXE 1

LISTE DES COMMANDES DE FLOPE

```
**********************************************************************
*                                                                    *
* L'espace de travail de l'éditeur contient  2 pointeurs :           *
*                                                                    *
*      COURANT  : le bloc (qui est mise en surbrillant)              *
*                 dans l'arbre de travail                            *
*                 sur lequel les commandes sont appliquées.          *
*                                                                    *
*      PILE     : contient la liste des blocs qui ne sont pas        *
*                 dans l'arbre de travail.                           *
*                                                                    *
**********************************************************************
```

A : ATTACHER

Cette commande permet de mettre le bloc en sommet de la PILE à coté du bloc COURANT.

H : en Haut (l'interface créée est horizontale)
B : en Bas (l'interface créée est horizontale)
D : à Droite (l'interface créée est verticale)
G : à Gauche (l'interface créée est verticale)

Le bloc qui vient d'être attaché devient le bloc COURANT.

B : BUFFER (manipulation)

S : Selection d'un buffer
L : Lister tous les buffers existants
C : Lister le buffer courant

C : CREATE

Cette commande permet de créer un bloc en utilisant (appelant) un module de prédiction topologique qui existe : ROM, PLA, LOGIQUE ALEATOIRE ou un autre module défini par l'utilisateur.

Le bloc créé peut être placé dans la PILE, ou directement à coté du bloc COURANT (dans ce cas, le bloc créé devient le bloc COURANT)

H : en Haut
B : en Bas
D : à Droite
G : à Gauche
P : au sommet de la Pile

Le module définit par l'utilisateur doit être déclaré en utilisant la macro "defmodule".

```
(defmodule
    (nature  "nature du module")
    (niveau  " mou ou flou ou dur")
    (entrées "liste de paramètres")
    (sorties "liste de paramètres")
    (contrainte "liste de paramètres")
    (programme  "nom de programme d'évaluation")
)
```

D : DETACHER

Détacher le bloc COURANT et l'empiler au sommet de la PILE ou supprimer le bloc du sommet de la PILE.
Le frère-avant devient le bloc COURANT.

C : détacher le bloc COURANT
P : supprimer le bloc qui est au sommet de la PILE

E: EDIT

Editer un fichier qui contient la description des blocs décrits sous forme interne ou externe, et puis l'empiler.

I : fichier sous forme Interne
E : fichier sous forme Externe

K : COPIER

O : Copier le bloc COURANT et l'empiler.

L : LISTER

Lister la description du bloc COURANT ou de tous les blocs du PLAN

C : Courant
P : Plan

On peut demander les paramètres à lister
n : nom du bloc
nt: nature du bloc
x : abscisse du bloc
y : ordonnée du bloc
dx: dimension suivant X
dy: dimension suivant Y
t : type de la transformation
r : type de l'interface
d : type de la déformabilité

M : MODIFICATION

Cette commande permet de modifier les dimensions du bloc COURANT en ré-utilisant (appelant) un module de prédiction topologique.

D : Dimension par un module de prédiction topologique
S : Dimension par la prédiction Standard
(en utilisant la courbe HYPERBOLE)
N : Nom du bloc

N : MARQUER

M : Mettre une marque sur le bloc COURANT
S : Sélectionner un bloc marqué
D : Detruire une marque
L : Lister les marques qui existent

O : OPTION

Cette commande permet de sélectionner les Options des visualisation.

F : dessiner en Full
V : dessiner en Vide
E : initialise l'échelle de la fenêtre de dessin

P : REPETITION

Cette commande permet de faire une répétition du bloc COURANT dans toutes les directions. Le résultat sera placé à coté du bloc COURANT. Cette commande est équivalente à une sequence de commande COPIE et ATTACHER.

H : répétition vers le Haut
B : répétition vers la Bas
D : répétition à Droite
G : répétition à Gauche

Q : QUIT

Sortir de l'éditeur.

R : ROTATION

Cette commande permet de faire une rotation de 90 degrés ou de -90 degrés du bloc COURANT. Après une opération de rotation le point origine du bloc reste le même.

N : rotation -90 degrés
(Négative : dans le sens des aiguilles d'une montre)
P : rotation 90 degrés
(Positive : dans le sens contraire)

S : SELECTION

Cette commande permet de faire la sélection d'un bloc en se déplacant dans la hiérarchie, à partir du bloc COURANT.

P : sélectionner le bloc Père
F : sélectionner le bloc Fils
A : sélectionner le bloc frère-Avant (à gauche ou en bas)
S : sélectionner le bloc frère-Suivant (à droite ou en haut)

T : sélectionner le bloc racine (Tête)
G : sélectionner le bloc précédemment sélectionné.
N : sélectionner un bloc par son nom

Pour les sélections par P, F, A et S on peut donner le numéro (par défaut il est égal à 1).

NOTE: on peut également utiliser les touches claviers ci-dessous pour la sélection.

<- equivalence à SA
-> equivalence à SS
↑ equivalence à SP
↓ equivalence à SF

T : TRANSLATION

Cette commande permet de translater le bloc COURANT. L'éditeur donne la valeur maximale permise pour la translation.

H : translation vers le Haut
B : translation vers le Bas
D : translation à Droite
G : translation à Gauche

U : SUBSTITUTION

Cette commande permet de substituer le bloc COURANT par le bloc qui se trouve en sommet de la PILE.

V : VISUALISATION

Cette commande permet de visualiser le PLAN (tous les blocs) ou seulement le bloc COURANT

P : visualisation du PLAN
C : visualisation du bloc COURANT

Z : SYMETRIE

Cette commande permet de faire un miroir du bloc COURANT par rapport à l'axe de X ou de Y.

Après une opération de symétrie le point origine du bloc reste le même.

X : par rapport à l'axe de X
Y : par rapport à l'axe de Y

ANNEXE 2

DECLARATION DES EVALUATEURS

Nous décrivons la déclaration des évaluateurs existants (prototype) dans le système CAPRI. Ces sont les évaluateurs de
- PLA [REI-83]
- ROM [REI-83], [MAI-83]
- LOGIQUE ALEATOIRE [REI-83].

```
(defmodule
   (program /users/syc/supri/modules/PLA.11 PLA)
   (nature PLA)
   (niveau flou)
   (entrees
       (ne |nombre d'entrées|)
       (nm |nombre de monomes|)
       (nn |nombre de niveaux|)
   )
   (contraintes
       tr    ;       taux de remplissage
       td    ;       taux de duplication des monomes
       pe    ;       pas d'entrée
       ps    ;       pas de sortie
       type  ;
   )
   (sorties
       dx
       dy
       doux    ; dimension X de matrice OU
       douy    ; dimension Y de matrice OU
       detx    ; dimension X de matrice ET
       dety    ; dimension Y de matrice ET
       dyt
       smet
       smou
   ) )


(defmodule
   (program /users/syc/supri/modules/LOGAL.11 LOGAL)
   (nature LOGAL)
   (niveau flou)
   (entrees
     (ntran |nombre de transistors|)
   )
   (sorties
      dx
      dy
      sbloc     ; surface de bloc
   )
   (contraintes
      nl
      kop
      cdx
      cdy
   ) )
```

```
(defmodule
   (program /users/syc/suprí/modules/ROM.11  ROM)
   (nature ROM)
   (niveau flou)
   (entrees
      (nb |nombre de bits par mot|)
      (nm |nombre de mots|)
   )
   (sorties
      dx
      dy
      ne
      ns
      stb
      st
   )
   (contraintes
      pe
      ps
      cdx
      cdy
   ) )

(defmodule
  (program /users/syc/suprí/modules/mailhot/ROM1.11  ROM1)
  (nature ROM1)
  (niveau flou)
  (entrees
     (nm |nombre de mots|)
     (nb |nombre de bits/mot|)
  )
  (sorties
     dx
     dy
     st
     stb
     ne
     ns
     NIV-DEC
     NIV-MUX
     NIV-SOR
  )
  (contraintes
     cdx
     cdy
     pe
     ps
     POS-DEC
     POS-MUX
     POS-SOR
  ) )
```

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974

VU les rapports de présentation de Messieurs

. F. ANCEAU, Professeur en disponibilité
. C. MASSON, Ingénieur

Monsieur SUWARDI Iping Supriana

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Fait à Grenoble, le 22 mai 1985

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

RESUME

Le travail présenté dans cette Thèse porte sur le domaine de l'aide à la construction du plan de masse de circuits VLSI. Cette construction est basée sur une évaluation topologique prédictive et une approche hiérarchisée.

FLOPE est un éditeur interactif permettant la construction d'un plan de masse de manière structurée.

Il est essentiellement destiné à communiquer avec des évaluateurs existants ou à venir.

Son rôle dans la conception hiérarchique est notamment :

- d'anticiper les problèmes de **composition** grâce à l'évaluation prévisionelle de surface, de forme et d'interconnexions lors de l'étape de **décomposition.**
- d'absorber souplement les modifications topologiques grâce à un mécanisme de **propagation.**

FLOPE a été implanté en langage CEYX-Le_Lisp.

Mots-clés :

éditeur hiérarchique, élaboration du plan de masse, évaluateur, cohérence topologique, propagation, voisinage, interface.